UNICHECK

Власник документу:
Попенко Володимир Дмитрович

Дата перевірки:
14.06.2020 01:14:14 EEST

Дата звіту:
14.06.2020 19:27:30 EEST

ID перевірки:
1004022438

Тип перевірки:
Doc vs Internet + Library

ID користувача:
77149

Назва документу: Mischenko_ip63

ID файлу: 1004035505 **Кількість сторінок:** 48 **Кількість слів:** 7390 **Кількість символів:** 55369 **Розмір файлу:** 128.28 KB

11.2% Схожість

Найбільша схожість: 3.15% з джерело бібліотеки. **ID файлу:** 1003962803

7.08% Схожість з Інтернет джерелами	115	Page 50
9.77% Текстові збіги по Бібліотеці акаунту	521	Page 52

4.3% Цитат

Цитати	1	Page 53

Вилучення переліку посилань вимкнено

0% Вилучень

Вилучений текст відсутній

Підміна символів

Заміна символів	1

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет інформатики та обчислювальної техніки

(повне найменування інституту, факультету)

Автоматизованих систем обробки інформації і управління

(повна назва кафедри)

«До захисту допущено»
В.о. завідувача кафедри
_____ *Олександр ПАВЛОВ*
(підпис) (ініціали, прізвище)
“ ” 2020 р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Програмне забезпечення інформаційних управляючих систем та технологій»

спеціальності «121 Інженерія програмного забезпечення»

на тему *Веб застосування для генерації питань за текстовими даними*

Виконав: студент IV курсу, групи *ІП-63 Міщенко Олександр Олексіович* _____
(прізвище, ім’я, по батькові) (підпис)

Керівник *доц., к.т.н., Іванова Л.М.* _____
посада,науковий ступінь,вчене звання,прізвище,і ім’я, по батькові (підпис)

Консультант з графічної документації *доц., к.т.н., Ліщук К.І.* _____
посада,науковий ступінь,вчене звання,прізвище,і ім’я, по батькові (підпис)

Рецензент:

ст. викладач, к.т.н., Дьяков С. О. _____
посада,науковий ступінь,вчене звання,прізвище,ім’я, по батькові (підпис)

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2020 року

Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”

Факультет (інститут) *Інформатики та обчислювальної техніки*
(повна назва)

Кафедра *автоматизованих систем обробки інформації і управління*
(повна назва)

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – *121 Інженерія програмного забезпечення*

Освітньо-професійна програма – *Програмне забезпечення інформаційних управляючих систем та технологій*

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри
_____ *Олександр ПАВЛОВ*
(підпис)
“___” _____2020 р.

З А В Д А Н Н Я
НА ДИПЛОМНИЙ ПРОЄКТ СТУДЕНТУ

Міщенко Олександру Олексійовичу
(прізвище, ім’я, по батькові)

1. Тема проєкту «Веб застосування для генерації питань за текстовими даними»
керівник проєкту *Іванова Любов Миколаївна доц., к.т.н.,*
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
затверджені наказом по університету від “07” травня 2020 р. №1081-с

2. Термін подання студентом проєкту «08» *червня 2020 року*

3. Вихідні дані до проєкту
Технічне завдання

4. Зміст пояснювальної записки
1) Аналіз вимог до програмного забезпечення: загальні положення, змістовний опис і аналіз предметної області, аналіз успішних IT-проєктів, аналіз вимог до програмного забезпечення
2) Моделювання та конструювання програмного забезпечення: моделювання та аналіз програмного забезпечення, архітектура програмного забезпечення, конструювання програмного забезпечення, аналіз безпеки даних
3) Аналіз якості та тестування програмного забезпечення: вступ та призначення плану тестування, предмети тестування, функціонал що підлягає тестуванню, функціонал що не підлягає тестуванню, підхід до тестування, критерії проходження тестування, критерії припинення тестування, вимоги до

середовища, опис тестів

4) Впровадження та супровід програмного забезпечення: розгортання програмного забезпечення, робота з програмним забезпеченням

5. Перелік графічного матеріалу

1) Схема структурна варіантів використання системи

2) Схема структурна класів програмного забезпечення

3) Креслення вигляду екранних форм

6. Консультанти розділів проєкту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання «10» *березня 2020 року*

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1.	*Вивчення рекомендованої літератури*	*16.03.2020*	
2.	*Аналіз існуючих методів розв'язання задачі*	*21.03.2020*	
3.	*Постановка та формалізація задачі*	*24.03.2020*	
4.	*Аналіз вимог до програмного забезпечення*	*30.03.2020*	
5.	*Алгоритмізація задачі*	*07.04.2020*	
6.	*Моделювання програмного забезпечення*	*13.04.2020*	
7.	*Обґрунтування використовуваних технічних засобів*	*16.04.2020*	
8.	*Розробка архітектури програмного забезпечення*	*19.04.2020*	
9.	*Розробка програмного забезпечення*	*30.04.2020*	
10.	*Налагодження програми*	*11.05.2020*	
11.	*Виконання графічних документів*	*17.05.2020*	
12.	*Оформлення пояснювальної записки*	*24.05.2020*	
13.	*Подання ДП на попередній захист*	*28.05.2020*	
14.	*Подання ДП рецензенту*		
15.	*Подання ДП на основний захист*	*08.06.2020*	

Студент _____ Олександр МІЩЕНКО
(підпис)

Керівник _____ Любов ІВАНОВА
(підпис)

№з/п	Формат	Позначення	Найменування	Кількість листів	Примітка
1	А4		Завдання на дипломний проєкт	2	
2	А4	КПІ.ІП-6319.045440.01.81	Пояснювальна записка	64	
3	А4	КПІ.ІП-6319.045440.02.91	Технічне завдання	11	
4	А4	КПІ.ІП-6319.045440.03.13	Опис програми	47	
5	А4	КПІ.ІП-6319.045440.04.51	Програма та методика тестування	6	
6	А4	КПІ.ІП-6319.045440.05.34	Керівництво користувача	19	
7	А3	КПІ.ІП-6319.045440.06.99.СС	Схема структурна варіантів використання системи	1	
8	А3	КПІ.ІП-6319.045440.07.99.КЕ	Креслення вигляду екранних форм	1	
9	А3	КПІ.ІП-6319.045440.08.99.СС	Схема структурна класів програмного забезпечення	1	

Зм.	Арк.	ПІБ	Підп.	Дата	КПІ.ІП-6319.045440.00.90		
Розробн.		Міщенко О.О.			Відомість дипломного проєкту	Літ.	Лист 1 / Листів 1
Керівн.		Іванова Л.М.					
Консульт						КПІ ім.Ігоря Сікорського кафедра АСОІУ гр. ІП-63	
Н/контр.		Ліщук К.І.					
В.о.зав.каф.		Павлов О.А.					

Пояснювальна записка
до дипломного проєкту

на тему: *Веб застосування для генерації питань за текстовими даними*

Київ – 2020 року

АНОТАЦІЯ

Пояснювальна записка до дипломного проєкту: 14 рис., 44 табл., 3 додатка, 10 джерел.

Об'єкт дослідження: процес автоматичної генерації тестових питань з варіантами відповідей на основі тексту.

Мета дипломного проекту: розробка засобів автоматизації процесу генерації тестових питань з варіантами відповідей на основі тексту.

У першому розділі проведено аналіз предметної області та вимог до програмного забезпечення. Були розроблені функціональні вимоги та діаграма функціональних вимог, діаграма варіантів використання та їх детальний опис, нефункціональні вимоги та матриця трасування.

У другому розділі описані архітектура, діаграма класів, специфікація методів, схема бази даних та таблиць, специфікація полів, використані підходи до написання програми та аналіз безпеки даних.

У третьому розділі представлено розроблений план тестування та результати самого тестування.

У четвертому розділі наведено, яким чином розгортати дане веб застосування. Також представлена інструкція користувача і інструкція програміста.

ТЕКСТ, ПИТАННЯ, ТЕСТ, ВАРІАНТИ ВІДПОВІДЕЙ, АЛГОРИТМ, NLP КОРПУС, NER МОДЕЛЬ, МОРФОЛОГІЧНА МОДЕЛЬ, ВЕБ ЗАСТОСУВАННЯ

ABSTRACT

Explanatory note to the degree project: 14 fig., 44 tab., 3 applications, 10 references.

The object of study: the process of automatic generation test text-based questions with answer options.

The aim of the degree project: development of automated agent, capable of performing the process of generation test text-based questions with answer options.

In the first section domain and software requirements were analyzed. Functional requirements and their diagram, chart use cases and their detail description, nonfunctional requirements and traceability matrix were developed.

In the second section program architecture, class diagram, methods specification, database schema, fields specification, approaches to developing software and analysis of safety data were described.

In the third section test plan and the results of the testing were submitted.

In the fourth section presents how to deploy this web application. There are user and programmer manual were described.

TEXT, QUESTIONS, TEST, ANSWER OPTIONS, ALGORITHM, NLP CORPORA, NER MODEL, MORPHOLOGICAL MODEL, WEB APPLICATION

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

NLP – технологія для обробки та аналізу тексту (англ. Natural Language Processing).

Word2vec – технологія, яка призначена для отримання слів у вигляді векторів з набором певних характеристик, виражених числовими параметрами (англ. Words to vectors).

NER – технологія, яка здатна знайти і класифікувати іменовану сутність (імена людей, організації, місця розташування, дати і т.д.), згадану в неструктурованому тексті (англ. Named Entity Recognition).

NLP корпус – модель, яка здатна знайти найбільш схожі слова на основі технології Word2vec та їх векторного представлення (англ. Natural Language Processing corpora).

pymorphy2 – бібліотека з морфологічним аналізатором, який застосовує словники з відкритих корпусів текстів для морфологічного та синтаксичного аналізу слів у реченні.

deeppavlov – бібліотека, яка надає доступ до різноманітних NLP моделей для аналізу речень у тексті.

IDE – система програмних засобів, яка використовується програмістами для розробки програмного забезпечення (англ. Integrated Development Environment).

ООТВ – особливість комп’ютерних програми або апаратної реалізації без установки будь-яких апаратних чи програмних розширень, функція яка надається буквально «з коробки» (англ. Out of the box).

API – прикладний програмний інтерфейс, опис способів на основі яких одна програма може взаємодіяти з іншою програмою (англ. application programming interface).

REST – підхід до архітектури мережевих протоколів, на основі які забезпечуються доступи до інформаційних ресурсів (англ. Representation State Transfer).

UI – засіб зручної взаємодії користувача з інформаційною системою (англ. User Interface).

UX — зручність, що людина відчуває при користуванні сервісом Основними об'єктами дослідження є враження, емоції та користь, отримані від взаємодії з продуктом. (англ. User Experience).

БД – база даних (англ. database).

СКУД – сукупність програмних і лінгвістичних засобів загального або спеціального призначення, для забезпечення управління створенням і використанням баз даних (англ. Database Management System, DBMS).

SQL – декларативна мова програмування для взаємодії користувача з базами даних, застосовується для формування запитів, оновлення і керування даними у реляційних БД, створення схеми БД і її модифікації, системи контролю за доступом до БД (англ. Structured query language).

MVC – архітектурний шаблон, який використовується під час проектування та розробки програмного забезпечення (англ. Model-view-controller).

HTTP – протокол передачі даних, що використовується в комп'ютерних мережах (англ. Hyper Text Transfer Protocol).

HTTPS – розширення протоколу HTTP з підтримкою шифрування з метою підвищення безпеки (англ. Hyper Text Transfer Protocol Secure).

UML – уніфікована мова моделювання (англ. Unified Modeling Language).

Salt – сіль паролю, рядок даних, який передається до хеш-функції разом з вхідним масивом даних для обчислення хешу.

ОЗП – оперативна пам’ять комп’ютера.

JSON – текстовий формат обміну даними між комп'ютерами (англ. JavaScript Object Notation).

Bootstrap – веб фреймворк з набором інструментів для швидкого та зручного створення веб сайтів та сервісів.

ВСТУП

Система освіти у будь-якому навчальному закладу потребує проведення контролю засвоєння знань у учнів та студентів. Але не завжди є можливість виділити час для детального аналізу рівня засвоєної інформації кожної людини, щоб детально оцінити індивідуальні слабкі місця у засвоєному матеріалі. Тому протягом кожного курсу/предмету у навчальних закладах проводяться контрольні роботи, колоквіуми та екзамени які допомагають сформувати оцінку відповідно до рівня знань кожного студента. І один з найпопулярніших типів питань у цих роботах незалежно від сфери наукової області – тести.

Тестові завдання легко вилучити з інформації у підручниках, статтях та лекціях, достатньо швидко перевіряються викладачами, тому дозволяють в досить короткий час провести перевірки знань у групи учнів. Але створення тестів вимагає від людини високого ступеня заглибленості в зміст тексту, а створення неправильних відповідей до тестового завдання дуже часто потребує додаткового часу на пошук схожих варіантів відповідей у цьому ж тексті або в інтернеті. Тому питання про можливість автоматично генерувати безліч тестових питань на будь-які теми для перевірки знань учнів без участі викладачів дуже гостро обговорюється у навчальних закладах.

Також у зв'язку з пандемією коронавіруса освітні заклади країн усього світу перейшли на електроне викладання предметів, і тести стали основною формою контролю процесу навчання. Вони легко створюються та швидко перевіряються викладачами, але що найважливіше - їх без особливих зусиль можна адаптувати під різноманітні системи тестування знань учнів, (наприклад у Google Classes, Quizlet, Classmarker та інш.), що дозволяють проводити своєчасний контроль знань у режимі онлайн, автоматично перевіряти результати у необмеженого числа зданих робіт відповідно до заздалегідь введеним викладачем відповідям та проставляти оцінки у віртуальних класах.

Завданням даної роботи є дослідження та вибір оптимальних існуючих алгоритмів комп’ютерної обробки текстів та створення питальних речень на основі тексту за довільною темою.

Результатом роботи є система, яка на основі введеного тексту генерує набір тестових завдань різноманітного формату з можливими варіантами відповідей стосовно теми речення, на базі якого генерується питання.

1 АНАЛІЗ ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

1.1 Загальні положення

На сьогоднішній день у майже усіх технологічних рішеннях, де потрібна постійна взаємодія с користувачем, реалізована спілкування машини з людиною. У кожному онлайн чаті служби підтримки є бот, який здатен адекватно підтримувати діалог без участі живого оператора; голосові помічники у смартфонах та комп'ютерах, такі як Siri, Google, Cortana, здатні спілкуватися, розпізнавати команди людини та виконувати їх – все це приклади машинного аналізу та комп'ютерної обробки текстів. Але як можна навчити машину розуміти, про що говорить людина, та надати їй можливість відповідати? Цю проблему вирішує технологія NLP – Natural Language Processing або природна обробка мови.

Це підрозділ інформатики, основна мета якого – робити детальний аналіз людської мови та представляти отримані дані у зрозумілий для машини формі. NLP застосовує алгоритми машинного навчання, які попередньо створюють та тренують на текстах, повідомленнях і розмовах людей, щоб отримати модель, яка здатна у повному обсязі розпізнавати будь-яку мову. Приклад важливих задач сучасного світу, яка вирішує ця технологія.

Машинний переклад з однієї мови на іншу. На сьогоднішній день є величезний прогрес, у кожної людини у смартфоні встановлено перекладач, який здатен за лічені долі секунд аналізувати величезні тексти та виконувати синтаксично правильну конвертацію у будь-яку мову. Але задача отримання повністю автоматичного перекладу найвищої якості з урахуванням всіх нюансів так поки залишається невирішеною. Приклад неточності перекладу наведений на рисунку 1.1.

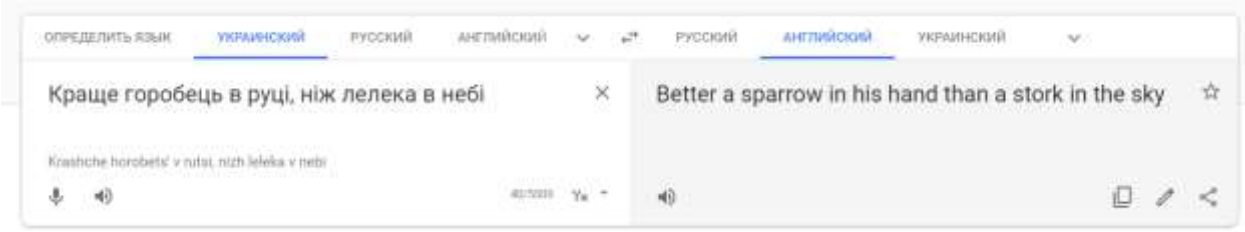


Рисунок 1.1 – Приклад перекладу української приказки

Інша задача - класифікація текстів за певними категоріями. Класичні варіанти, які вже реалізовані у більшості сучасних сервісах, це фільтрація новин за темами: політика, спорт, бізнес; відокремлення листів в електронній пошті по папкам та фільтрація їх спам; класифікація відгуків від користувачів на позитивні, нетральні та негативні та інші.

Також ця технологія застосовується для створення питально-відповідних діалогових систем, які широко застосовуються у чат-ботах. Наприклад, текстовий аналіз по окремими словам у реченні дозволяє зрозуміти, чи потрапляє користувач у сценарій запрограмованої відповіді бота з повідомленням «як відкрити кредитну карту».

Ще одна важлива сфера застосування NLP, без якої неможлива реалізація попередніх функцій, це вирішення задачі знаходження, аналізу та класифікації іменованих сутностей у тексті - NER (Named Entity Recognition). Без застосування NER технології неможливо уявити імплементацію наступних задач: вирішення анафори с займенниками у реченнях, побудова питальних систем у веб браузерах та звичайно пошук, сортування та фільтрація інформації у текстах по спеціальним іменованим сутностям.

Так як NLP технологія широко використовується у багатьох сферах на сьогоднішній день, то в інтернеті можна знайти готові та натреновані моделі на різноманітних текстових датасетах на будь-яку мову. Це може бути модель, яка орієнтується на новинні статті, сторінки з інформаційного порталу або пости з соціальних мереж. Але важливо ще те,

для чого була призначена створена модель. Це може бути модель, яка намагається класифікувати задачу користувача у вхідному реченні (наприклад, якщо на вхід подається речення «ввімкни трек “Never Gonna Give You Up”», то модель повинна розуміти, що це задача на музичний плеєр); модель, яка здатна заповнювати пропуски у реченні найбільш відповідними за змістом словами; модель, яка виконує морфологічний аналіз речення та встановлює відповідно до кожного слова його частину мови, рід, число, відмінок та інші параметри; модель, яка здатна формувати список найбільш часто вживаних словосполучень відповідно до вхідного слова та безліч інших.

На основі комбінування різноманітних NLP моделей реалізовані сервіси описані вище, якими кожен день користуються мільйони людей по всьому світу. Але на сьогоднішній день не існує реалізації програми, яка на основі NLP технології може генерувати тестові завдання та питання українською мовою базуючись на інформації у тексті.

Актуальність розробки обумовлена тим, що зараз тести – одна з основних складових частин оцінки якості освітніх результатів людини. Вони широко використовуються у школах, університетах, на курсах і т.д, їх легко перевіряти та швидко можна адаптувати під сервіси, які реалізують функції електронного навчання. Тому розробка сервісу, що здатен облегшити цей процес, допоможе багатьом викладачам у процесі створення завдань до свого предмету.

Використання сервісу для автоматичної генерації тестових питань на основі книг, лекцій, статей дозволяє суттєво зменшити витрати часу на підготовку тестів для контролю знань у освітніх закладах, бо просте створення питань вимагає від людини високого ступеня заглибленості в зміст тексту. А якщо це створення тестових завдань з декількома варіантами відповідей, то ще необхідно витрачати час на їх підготовку, бо неякісні можливі варіанти відповідей збільшують ймовірність вгадування.

В рамках даної дипломної роботи було використано три моделі, які були створені на основі усіх українських статей з Вікіпедії: NER модель для аналізу іменованих сутностей, морфологічна модель для аналізу слів у реченні та NLP корпус, який здатний шукати найбільш часто вживані словосполучення відповідно до вхідного слова.

1.2 Змістовний опис і аналіз предметної області

Генерація питання на основі тексту складається з трьох основних етапів, які вказані на рисунку 1.2.

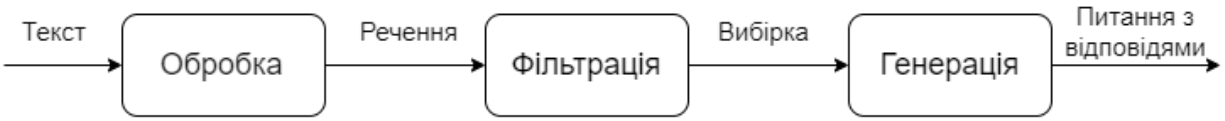


Рисунок 1.2 – Процес роботи алгоритму

Перший етап - попередня обробка тексту та його розбиття на окремі речення, на інформації з котрих потім будуть генеруватися тестові завдання. Основна складність полягає у визначенні початку та кінця речення, так як текст може містити комбінації з пунктуаційних знаків, різних абревіатур та скорочень. Для вирішення даної проблеми застосовуються регулярні вирази та заздалегідь створений стандартний перелік скорочень (наприклад чол., га., куб. та інші).

Другий етап – знаходження цікавих фактів у створених реченнях, до яких можна поставити питання. Це може бути ім'я видатної людини, яка зробила наукове відкриття, історична дата відомої події, назва країни, міста або географічного об'єкта. Для того, щоб вирішити цю задачу, застосуємо NER модель, навчену на українських статтях з Вікіпедії.

NER технологія використовується для вирішення проблеми розпізнання та сортування іменованих об'єктів у тексті. Для того щоб здійснити пошук, метою якого є класифікація іменованих об’єктів в тексті по певним категоріям,

таким як імена людей, організації, місця розташування, вираження часу, кількості, грошові значення, відсотки і т. д., були розроблені NER моделі. Для створення моделі для аналізу певної мови використовують методи лінгвістичної граматики цієї мови та статистичні моделі, такі як машинне навчання.

Приклад аналізу іменованих сутностей речення «Старт корабля “Восток” з пілотом-космонавтом Юрієм Олексійовичем Гагаріним на борту був проведений 12 квітня 1961 року о 09:07 за московським часом з космодрому Байконур.» наведено у таблиці 1.1.

Таблиця 1.1 – Приклад аналізу іменованих сутностей у реченні

Номер	**Слово**	**Тег**
1	Старт	O
2	корабля	O
3	“	B-PRODUCT
4	Восток	I-PRODUCT
5	”	I-PRODUCT
6	з	O
7	пілотом	O
8	-	O
9	космонавтом	O
10	Юрієм	B-PERSON
11	Олексійовичем	I-PERSON
12	Гагаріним	I-PERSON
13	на	O
14	борту	O
15	був	O
16	проведений	O
17	12	B-DATE

Продовження таблиці 1.1

Номер	**Слово**	**Тег**
18	квітня	I-DATE
19	1961	I-DATE
20	року	I-DATE
21	о	O
22	09	B-TIME
23	:	I-TIME
24	07	I-TIME
25	за	I-TIME
26	московським	I-TIME
27	часом	I-TIME
28	з	O
29	космодрому	O
30	Байконур	B-GPE
31	.	O

Про що говорить встановлений тег відповідно до кожного слова та пунктуаційного знаку? Якщо тег «O», то аналізоване слово не належить до іменованої сутності. Знак «B-» перед тегом вказує на початок нового знайденого об’єкту, це зроблено для відокремлення двох різних іменованих сутностей, котрі стоять один після одного у реченні. У аналізованому прикладі реченні знайшлись такі об’єкти: сутність імені продукту, людини, дата, час та локація.

Після обробки створених речень NER моделлю, отримаємо список іменованих сутностей у кожному реченні. Ці дані будуть використовуватися у майбутньому як відповіді для створених питань. Але для того, щоб граматично правильно поставити питання до речення, треба знати частини мови слів у реченні, їх відмінки та зв'язок один з одним. Для того, щоб вирішити цю

задачу, застосуємо морфологічну модель, також навчену на статтях з української мови.

Морфологічні моделі також створюються за допомоги машинного навчання за допомоги фреймворку Universal Dependencies, який містить інформацію стосовно 122 тисяч слів української мови та 27 тисяч речень з прикладами їх використання. На виході після створення нейромережі отримуємо морфологічну модель , яка здатна виконувати аналіз кожного слова на предмет частини мови, знаходити лему кожного слова та встановлювати специфічні характеристики, наприклад рід, число, відмінок. Також варто відзначити, що ця модель на відміну від бібліотеки pymorphy2, яка теж реалізує цей функціонал, бере до уваги інші слова у реченні протягом обробки, тому для одного слова призначені частини мови можуть відрізнятися в залежності від контексту. Приклад морфологічного аналізу речення «Старт корабля “Восток” з пілотом-космонавтом Юрієм Олексійовичем Гагаріним на борту був проведений 12 квітня 1961 року о 09:07 за московським часом з космодрому Байконур.» наведено у таблиці 1.2.

Таблиця 1.2 – Морфологічний аналіз речення

Номер	**Слово**	**Лема**	**Частина мови**	**Додаткова інформація**
1	Старт	старт	NOUN	Animacy=Inan, Case=Nom, Gender=Masc, Number=Sing
2	корабля	корабль	NOUN	Animacy=Inan, Case=Gen, Gender=Masc, Number=Sing
3	“	“	PUNCT	-

Продовження таблиці 1.2

Номер	**Слово**	**Лема**	**Частина мови**	**Додаткова інформація**
4	Восток	восток	PROPN	Animacy=Inan, Case=Nom, Gender=Masc, Number=Sing
5	”	”	PUNCT	-
6	з	з	ADP	-
7	пілотом	пілот	NOUN	Animacy=Inan, Case=Ins, Gender=Masc, Number=Sing
8	-	-	PUNCT	-
9	космонавтом	космонавт	NOUN	Animacy=Inan, Case=Ins, Gender=Masc, Number=Sing
10	Юрієм	московський	PROPN	Animacy=Inan, Case=Ins, Gender=Masc, Number=Sing
11	Олексійовичом	час	PROPN	Animacy=Inan, Case=Ins, Gender=Masc, Number=Sing
12	Гагаріном	з	PROPN	Animacy=Inan, Case=Ins,

Продовження таблиці 1.2

Номер	**Слово**	**Лема**	**Частина мови**	**Додаткова інформація**
				Gender=Masc, Number=Sing
13	на	на	ADP	-
14	борту	борт	NOUN	Animacy=Inan, Case=Loc, Gender=Masc, Number=Sing
15	був	бути	AUX	Aspect=Imp, Gender=Masc, Mood=Ind, Number=Sing, Tense=Past, VerbForm=Fin, Voice=Act
16	проведений	проводити	VERB	Aspect=Perf, Gender=Masc, Number=Sing, Tense=Past, Variant=Short, VerbForm=Part, Voice=Pass
17	12	12	NUM	-
18	квітня	квітень	NOUN	Animacy=Inan, Case=Gen, Gender=Masc, Number=Sing

Продовження таблиці 1.2

Номер	**Слово**	**Лема**	**Частина мови**	**Додаткова інформація**
19	1961	1961	NUM	-
20	року	рік	NOUN	Animacy=Inan, Case=Gen, Gender=Masc, Number=Sing
21	о	о	ADP	-
22	09	09	NUM	-
23	:	:	PUNCT	-
24	07	07	NUM	-
25	за	за	ADP	-
26	московським	московський	ADJ	Case=Dat, Degree=Pos, Gender=Neut, Number=Sing
27	часом	час	NOUN	Animacy=Inan, Case=Dat, Gender=Masc, Number=Sing
28	з	з	ADP	-
29	космодрому	космодром	NOUN	Animacy=Inan, Case=Gen, Gender=Masc, Number=Sing
30	Байконур	Байконур	PROPN	Animacy=Inan, Case=Nom,

Продовження таблиці 1.2

Номер	**Слово**	**Лема**	**Частина мови**	**Додаткова інформація**
				Gender=Masc, Number=Sing
31	.	.	PUNCT	-

Морфологічний аналіз з обраної моделі дозволяє отримати лему кожного слова, його частину мови та додаткову інформацію, таку як відмінок (Case), стать (Gender), число (Number), одухотворення (Animacy), ступінь порівняння (Degree), час (Tense), форму (VerbForm), аспект (Aspect) та пасивна чи активна форма (Voice). У прикладі представлені наступні частини мови: іменники (NOUN, PROPN для власних імен), дієслова (VERB, AUX для службових та допоміжних), прикметники (ADJ), прийменники (ADP), числа (NUM) та знаки пунктуації (PUNCT). Ці дані будуть використовуватися у майбутньому для генерації питань, коли потрібно буде з речення видалити правильну відповідь та міняти слова місцями, щоб речення набуло форми питання. Таким чином, результат обробки дозволяє отримати всю необхідну інформацію про слова у реченні для побудови граматично та синтаксично правильного тестового питання.

Наступний крок – обробка створених та проаналізованих речень. В українській мові дуже багато займенників, і якщо людина може простежити взаємозв'язок базуючись на контексті, то алгоритм ні, адже він обробляє кожне речення не беручи до уваги попередні. Тому для кожного займенника знаходимо іменник у попередньому реченні за числом та статтю і робимо заміну, щоб не втратити майбутні можливі варіанти для генерації тестових завдань.

Другий етап – фільтрація речень, до яких може бути поставлене питання. Сервіс реалізовує два підходи: за участю користувача та без. Після першого

етапу користувач може відокремити спеціальні речення, ключові слова або налаштувати фільтр по конкретним типам слів з NER обробки для генерації питань. Якщо таких даних нема, то алгоритм підраховує кількість кожної іменованої сутності, які зустрічалась протягом тексту та генерує питання по найбільш часто згадуваним. Речення, до яких неможливо поставити питання до вибірки не потрапляють.

Останній етап – генерація питання з варіантами відповідей. Базуючись на результатах фільтрації з попереднього етапу обираємо об'єкт, до якого будемо ставити питання та записуємо його у відповідь. Далі генеруємо два типу питань: с заповненням пропусків та з питальним словом. Для питання першого типу замінюємо відповідь у реченні пропусками та на початок додаємо конструкцію типу «Вставте пропущені дати: ». Для питань другого типу робимо перестановку слів у речень за шаблоном:

$$Question = PREP(w) + QuestionWord(w, f) + VERB(w) + OtherWords$$

Тут w – слово або словосполучення, що є відповіддю на питання; $PREP(w)$ – прийменник від w, якщо він існує; $QuestionWord(w, f)$ – питальне слово в залежності від типу w та форми f; $VERB(w)$ – дієслово від w; $OtherWords$ – слова, які залишились у вхідному реченні.

Також з кожного питання робиться тестове завдання шляхом додавання 3 неправильних відповідей. Інші варіанти відповідей будуть вилучатись двома способами з вхідного тексту та NLP корпуса.

Перший варіант – пошук у тексті таких самих іменованих сутностей за встановленим тегом і формування додаткових варіантів відповіді на питання з знайдених об'єктів.

Другий варіант включає в себе використання NLP корпуса побудованому також на українських статтях з Вікіпедії. В процесі його створення на етапі аналізу купи текстів алгоритм запам'ятовує, які комбінації слів найбільш часто зустрічаються в одному контексті і на основі цих даних формує корпус з результатами. Це досягається за допомоги технології

Word2vec, суть якої полягає у представлені кожного слова у вигляді вектора з числами, де кожне число відповідає одній характеристиці слова. На основі даних у векторі слова-відповіді маємо можливість знайти похожі вектори за цією інформацією. Приклад візуалізації векторів з даними зображений на рисунку 1.2.

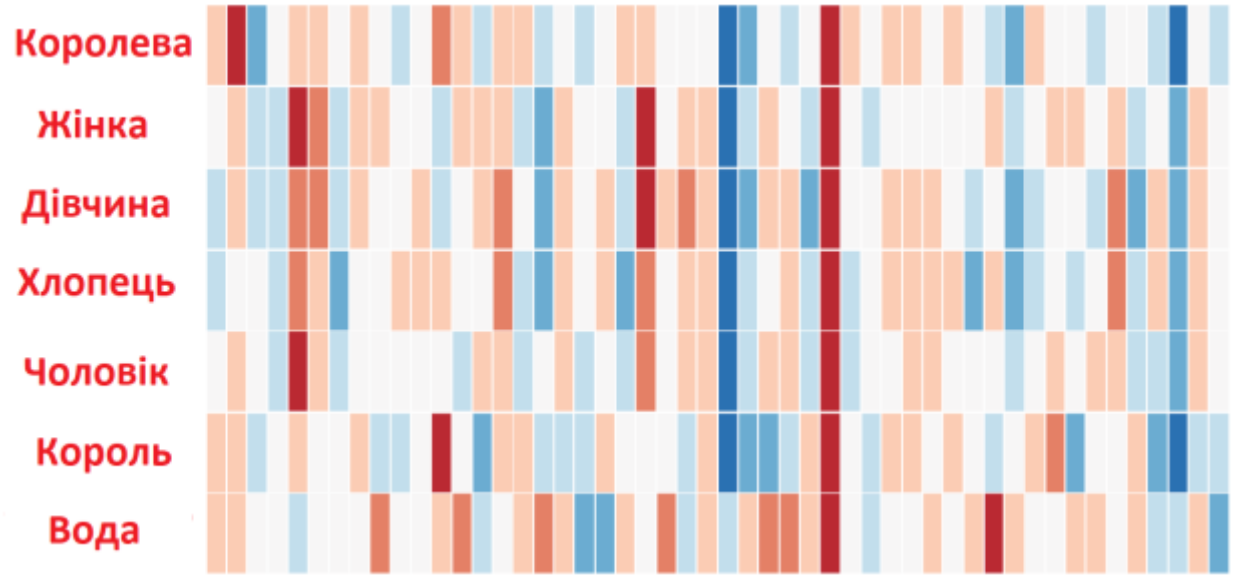


Рисунок 1.3 – Приклад векторних представлень слів

Можна помітити, що слова «жінка» і «дівчина» та слова «чоловік» та «хлопець» багато в чому схожі в деяких вимірах. На основі цих даних розраховується коефіцієнт наближеності до слова. Таким чином, для кожної відповіді, яка являється іменованою сутністю, можна знайти ще таких самих 1-3 варіанта відповіді, які будуть мати максимально схоже векторне представлення, такий самий NER тег, та використовуватися у схожому контексті.

Наприклад, для слова «Байконур» отримаємо таку вибірку найбільш суміжних слів:

- ('космодром_NOUN', 0.6827108860015869);
- ('казахстанський_NOUN', 0.665603518486023);
- ("плесецьк_NOUN', 0.6362855434417725);

- ('ракета_NOUN', 0.606155276298523);
- ('капустин::яр_NOUN, 0.5537377595901489).

Числа навпроти знайдених слів відображають ймовірність, що задане слово буде максимально схожим зі словом «Байконур». Після NER аналізу отриманих слів отримаємо, що «плесецьк_NOUN» та «капустин::яр_NOUN» мають такій ж самий NER тег, як і відповідь до питання, а це значить, що їх можна використовувати для створення неправильних тестових відповідей.

На виході отримаємо готове тестове завдання до речення, залишилось повторити алгоритм для всіх речень у тексті.

1.3 Аналіз успішних IT-проектів

1.3.1 Аналіз відомих технічних рішень

Для реалізації алгоритму генерації тестових питань необхідно використовувати три моделі: NER модель для аналізу іменованих сутностей та пошуку слів, до яких буде поставлене питання; морфологічна модель для аналізу слів, на базі яких синтаксично правильно буде відбуватися побудова питального речення та NLP корпус для пошуку найбільш часто вживаних варіантів словосполучень відповідно до відповіді для генерації тестових варіантів.

NER та морфологічна натреновані моделі були використані з бібліотеки deeppavlov. Це мультинаціональні моделі, які підтримають українську та російську мову. Через це ймовірність розпізнати іменовані сутності у реченні становить 88.8% з розміром 1.4 Гб , але на відміну від одномовних аналогів меншого розміру и наявністю більшої точності розпізнання, має можливість представляти іменовані сутності 18 видами тегів, а не 4. Морфологічна модель робить правильний аналіз слова у контексті з ймовірністю 97.8%, що є відміним показником, так як майже виключається ймовірність неправильного розпізнання частин мови, які потрібні стояти у питальному реченні на певних місцях. Але за таку точність

та можливість мультинаціонального аналізу доводиться також жертвувати розмірами моделі, вона важить 661 Мб. Можна підібрати та використовувати моделі значно меншого розміру відповідно до вимог розміру ОЗП сервера веб застосування, але тоді виключається можливість аналізувати тексти російською мовою.

Для генерації неправильних відповідей був використаний готовий та натренований NLP корпус з бібліотеки gensim. На основі технології векторного представлення слів маємо можливість шукати можливі варіанти відповідей з такими ж самими NER тегами та генерувати тестові завдання.

Цих трьох моделей достатньо, щоб генерувати різноманітні питання до наукових текстів.

1.3.2 Аналіз відомих програмних продуктів

В ході дослідження було знайдено два сервіси – Quillionz та ParaQG.

Quillionz – розроблений в Індії компанією Harbinger Group веб-сайт для генерації різноманітних питань, екзаменів та тестів англійською мовою.

Проект має наступні переваги: генерація True/False тестів, створення питань, де потрібно встановити відповідність (наприклад країни та їх столиці). Сервіс надає можливість подати вхідні дані для опрацювання у вигляді посилання на веб-сторінку та конвертувати аудіо- та відео-файли до тексту.

Але також цей сервіс має два суттєвих недоліки: для користування багатьма описаними вище функціями треба оформлювати платну місячну підписку на сайт. І також цей продукт не підтримує українську мову, що робить неможливим його використання у навчальних закладах України.

Приклади роботи зображені на рисунках 1.4 – 1.5.

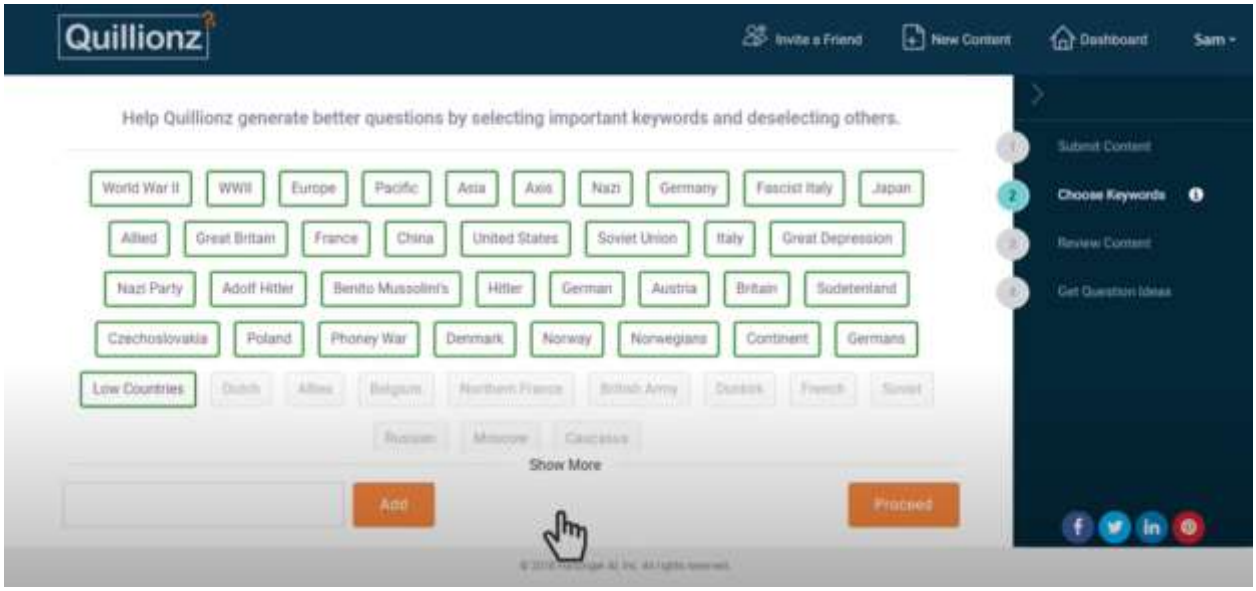


Рисунок 1.4 – Вибір іменованих сутностей для генерації питань

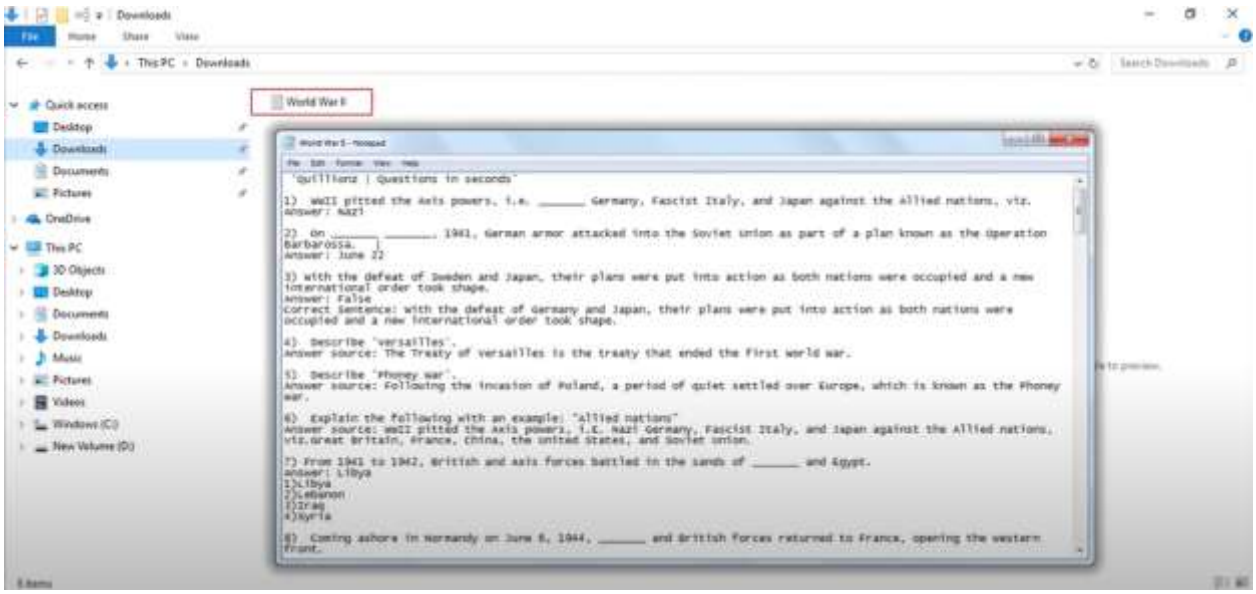


Рисунок 1.5 – Сгенеровані питання у текстовому файлі

Наступна розробка – ParaQG.

Основні переваги цього продукту: він повністю безкоштовний та реалізовує функцію представлення згенерованого питання в декількох формах с різними порядком слів, що надає можливість створювати питання з перемішаним порядком слів, що робить неможливим бистре знаходження правильної відповіді у тексті.

Але також сервіс має вагомі недоліки: генерація питань тільки за допомоги питальних слів, відсутність варіанту вибору правильної відповіді і відсутність можливості обробки тексту українською мовою.

Приклади роботи зображені на рисунках 1.6 – 1.7.

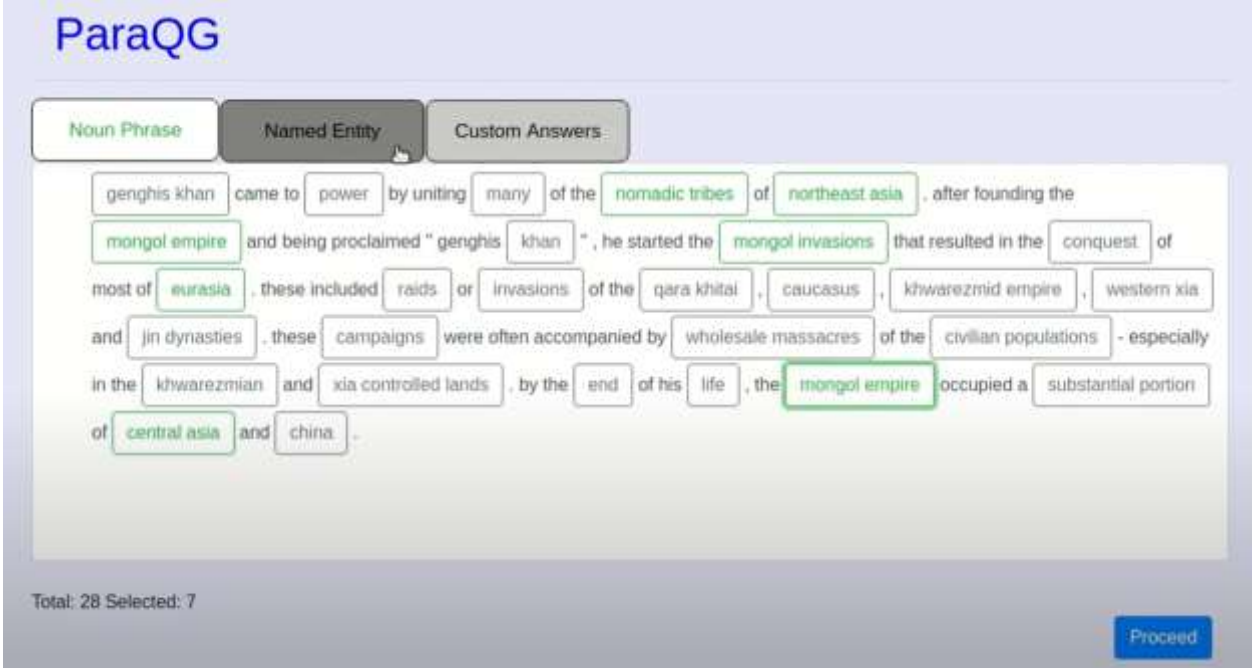


Рисунок 1.6 – Вибір іменованих сутностей для генерації питань

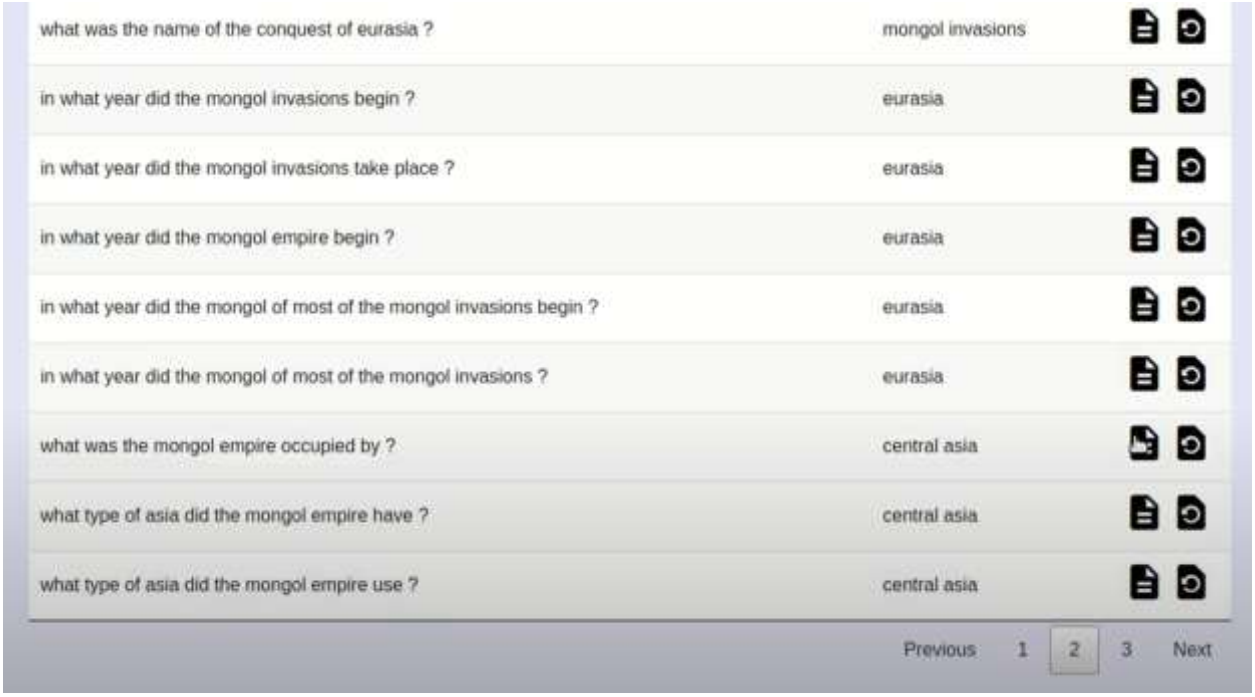


Рисунок 1.7 – Приклад сгенерованих питань

1.4 Аналіз вимог до програмного забезпечення

Користувачами системи можна виділити неавторизованого користувача та авторизованого користувача.

Сервіс має забезпечувати такі функції для неавторизованого користувача:

- реєстрація;
- авторизація;
- можливість введення тексту для опрацювання;
- можливість завантаження і конвертації документів формату txt, doc та pdf;
- можливість вибирати ключові слова для формування питань;
- можливість вибирати речення для формування питань;
- можливість вибирати теги для формування питань;
- можливість генерації питань по тексту;
- можливість експортувати результат у форматі txt, doc та pdf документів.

Сервіс має забезпечувати такі функції для авторизованого користувача:

- перегляд історії опрацьованих текстів та згенерованих питань;
- вихід з системи.

Система повинна надавати наступні варіанти використання, які описані в таблицях 1.3 – 1.13.

Таблиця 1.3 - Варіант використання UC001

Назва	Реєстрація в системі
Опис	Неавторизований користувача реєструється в системі
Учасники	Неавторизований користувач
Передумови	Відкрита сторінка реєстрації

Продовження таблиці 1.3

Постумови	Користувач успішно зареєструвався в системі
Основний сценарій	Користувач заповнює усі поля у формі (емейл, нік та пароль) валідними даними та натискає кнопку реєстрації
Розширення сценаріїв	Система повідомляє користувача про невалідні дані, це може бути занятий емейл фбо нік для реєстрації

Таблиця 1.4 - Варіант використання UC002

Назва	Авторизація в системі
Опис	Неавторизований користувач авторизується в системі
Учасники	Неавторизований користувач
Передумови	Користувач зареєстрований в системі. Відкрита сторінка авторизації
Постумови	Користувач успішно авторизувався в системі
Основний сценарій	Користувач заповнює поля (нік та пароль) у формі авторизіції валідними даними та натискає кнопку авторизації
Розширення сценаріїв	Система повідомляє користувача про невалідні дані

Таблиця 1.5 - Варіант використання UC003

Назва	Введення тексту
Опис	Користувач має можливість ввести текст для генерації питань
Учасники	Користувач

Продовження таблиці 1.5

Передумови	Відкрита сторінка генерації питань
Постумови	Оброблений текст
Основний сценарій	Користувач додає текст у форму та натискає кнопку аналізу
Розширення сценаріїв	Система повідомляє користувача про відсутність тексту, якщо була натиснута кнопка без введення

Таблиця 1.6 – Варіант використання UC004

Назва	Завантаження текстових файлів
Опис	Користувач має можливість завантажити текстові файли для конвертації формату txt, doc або pdf
Учасники	Користувач
Передумови	Відкрита сторінка генерації питань
Постумови	Файл був завантажений, текст був сконвертований та оброблений
Основний сценарій	Користувач завантажує файл та натискає кнопку аналізу
Розширення сценаріїв	Система повідомляє користувача про відсутність файлу, якщо була натиснута кнопка без завантаження

Таблиця 1.7 – Варіант використання UC005

Назва	Вибір ключових слів
Опис	Користувач має можливість обрати ключові слова для генерації питань
Учасники	Користувач
Передумови	Відкрита сторінка генерації питань, наявність обробленого тексту

Продовження таблиці 1.7

Постумови	Фільтри по ключовим словам встановлено
Основний сценарій	Користувач у формі обирає ключові слова для генерації питань
Розширення сценаріїв	

Таблиця 1.8 – Варіант використання UC006

Назва	Вибір речень
Опис	Користувач має можливість обрати речення для генерації питань
Учасники	Користувач
Передумови	Відкрита сторінка генерації питань, наявність обробленого тексту
Постумови	Фільтри по реченням встановлено
Основний сценарій	Користувач у формі обирає речення для генерації питань
Розширення сценаріїв	

Таблиця 1.9 – Варіант використання UC007

Назва	Вибір тегів
Опис	Користувач має можливість обрати теги для генерації питань
Учасники	Користувач
Передумови	Відкрита сторінка генерації питань, наявність обробленого тексту
Постумови	Фільтри по тегам встановлено
Основний сценарій	Користувач у формі обирає теги для генерації питань
Розширення сценаріїв	

Таблиця 1.10 – Варіант використання UC008

Назва	Генерація питань
Опис	Користувач має отримувати згенеровані питання
Учасники	Користувач
Передумови	Відкрита сторінка генерації питань, наявність обробленого тексту
Постумови	Отримання питань з відповідями по тексту
Основний сценарій	Користвач натискає кнопку генерації питань
Розширення сценаріїв	

Таблиця 1.11 – Варіант використання UC009

Назва	Конвертація згенерованих питань
Опис	Користувач має отримувати згенеровані питання у файлі формату txt, doc або pdf
Учасники	Користувач
Передумови	Відкрита сторінка генерації питань, наявність обробленого тексту, наявність згенерованих питань
Постумови	Формування файлу з даними
Основний сценарій	Користвач натискає кнопку конвертації до файлу
Розширення сценаріїв	

Таблиця 1.12 – Варіант використання UC010

Назва	Перегляд історії
Опис	Авторизований користувач переглядає історію роботи з сервісом
Учасники	Авторизований користувач
Передумови	Користувач авторизувався в системі, відкрита сторінка історії
Постумови	Відображення історії

Продовження таблиці 1.12

Основний сценарій	Користувач натискає кнопку перегляду історії роботи
Розширення сценаріїв	

Таблиця 1.13 – Варіант використання UC011

Назва	Вихід з системи
Опис	Авторизований користувач виходить з системи
Учасники	Авторизований користувач
Передумови	Користувач авторизувався в системі
Постумови	Відкрита сторінка авторизації
Основний сценарій	Користувач натискає кнопку виходу з системи
Розширення сценаріїв	

На основі проаналізованої моделі варіантів використання була побудована структурна схема використання. Схему можна побачити на у додатках до дипломної роботи.

Згідно аналізу вимог до програмного забезпечення було сформульовано ряд функціональних вимог. Функціональні вимоги наведені в таблицях 1.14 – 1.24.

Таблиця 1.14 – Опис функціональної вимоги REQ001

Номер	REQ001
Назва	Реєстрація в системі
Опис	Система має надавати незареєстрованому користувачеві можливість зареєструватися, заповнивши поля «Email», «Password» та «Confirm Password» та натиснувши на кнопку «Sign up»

Таблиця 1.15 – Опис функціональної вимоги REQ002

Номер	REQ002
Назва	Авторизація в системі
Опис	Система має надавати неавторизованому
Номер	користувачеві можливість авторизуватись у системі заповнивши поля «Login», «Password» та натиснувши кнопку «Sign in».

Таблиця 1.16 – Опис функціональної вимоги REQ003

Номер	REQ003
Назва	Введення тексту
Опис	Система має надавати користувачеві можливість введення текст

Таблиця 1.17 – Опис функціональної вимоги REQ004

Номер	REQ004
Назва	Завантаження текстових файлів
Опис	Система має надавати користувачеві можливість завантаження текстових файлів наступних форматів: txt, doc та pdf

Таблиця 1.18 – Опис функціональної вимоги REQ005

Номер	REQ005
Назва	Вибір ключових слів
Опис	Система має надавати користувачеві можливість фільтрувати речення за ключовими словами для генерації питань

Таблиця 1.19 – Опис функціональної вимоги REQ006

Номер	REQ006
Назва	Вибір речень
Опис	Система має надавати користувачеві можливість обирати речення для генерації питань

Таблиця 1.20 – Опис функціональної вимоги REQ007

Номер	REQ007
Назва	Вибір тегів
Опис	Система має надавати користувачеві можливість фільтрувати речення за тегами для генерації питань

Таблиця 1.21 – Опис функціональної вимоги REQ008

Номер	REQ008
Назва	Генерація питань
Опис	Система має генерувати питання відповідно до тексту

Таблиця 1.22 – Опис функціональної вимоги REQ009

Номер	REQ009
Назва	Конвертація згенерованих питань
Опис	Система має конвертувати згенеровані питання до текстових файлів наступних форматів: txt, doc та pdf

Таблиця 1.23 – Опис функціональної вимоги REQ010

Номер	REQ010
Назва	Перегляд історії

Продовження таблиці 1.23

Опис	Система має надавати авторизованому користувачеві можливість переглядати історію роботи з сервісом

Таблиця 1.24 – Опис функціональної вимоги REQ012

Номер	REQ012
Назва	Вихід з системи
Опис	Система має надавати авторизованому користувачеві можливість виходу з системи

Залежності між варіантами використання та функціональними вимогами показано на матриці трасування, що зображена на рисунку 1.8.

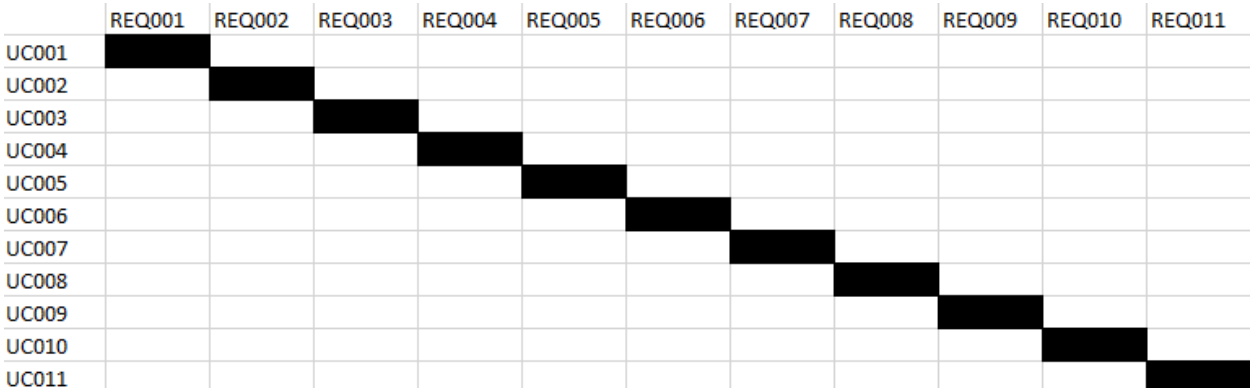


Рисунок 1.8 – Матриця трасування між варіантами використання та функціональними вимогами

1.4.1 Розроблення нефункціональних вимог

Була проведена розробка нефукціональних вимог до програмного забезпечення. Виявлені нефунціональні вимоги описані в таблицях 1.23 – 1.25.

Таблиця 1.23 – Опис нефункціональної вимоги NFR001

Номер	NFR001
Назва	Працездатність

Продовження таблиці 1.23

Опис	Веб-застосування повинно працювати тільки якщо компьютер має доступ до інтернету

Таблиця 1.24 – Опис нефункціональної вимоги NFR002

Номер	NFR002
Назва	Швидкодія
Опис	Обробка та генерація питань сторінки тексту не повинно тривати більш ніж 1000мс

Таблиця 1.25 – Опис нефункціональної вимоги NFR003

Номер	NFR003
Назва	Зручність і простота
Опис	Веб-застосування повинно бути зручним та інтуітивно зрозумілим для користувачів

1.4.2 Постановка комплексу завдань модулю

Згідно з розробленими функціональними та нефункціональними вимогами метою розроблення даного сервісу є наступний перелік комплексу завдань:

- надати можливість користувачу створювати обліковий запис;
- надати можливість користувачу завантажувати текст у різних форматах;
- надати можливість користувачу робити фільтр по речення для створення питань;
- надати можливість користувачу перегляди історію роботи з сервісом.

1.5 Висновки по розділу

У даному розділі було проведено змістовний аналіз та опис предметної області, були розглянуті технічні рішення. Також було проведено детальний аналіз існуючих аналогів, що вирішують схожий набір задач. Було проведено порівняння з розроблюваною системою та виявлені недоліки та переваги у порівнянні з конкурентами та виявлено, що існуючи сервіси не надають засобів для аналізу текстів українською мовою.

Також була наведені таблиці функціональних та нефункціональних вимог та поставлений комплекс завдань для розроблюваного модулю.

2 МОДЕЛЮВАННЯ ТА КОНСТРУЮВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Моделювання та аналіз програмного забезпечення

Для створення програмного забезпечення необхідно спроектувати бізнес-процеси. Для їх проектування було обрано методологію BPMN (Business Process Model and Notation) діаграм. У ході аналізу та моделювання програмного забезпечення, було виділено декілька основних бізнес-процесів:

- реєстрація у системі;
- авторизація у системі;
- генерація питань;
- перегляд історії використання;

Відповідні BPMN діаграми наведено нижче на рисунках 2.1-2.4.

Рисунок 2.1 – Схема бізнес-процесу реєстрації

Рисунок 2.2 – Схема бізнес-процесу авторизації

Рисунок 2.3 – Схема бізнес-процесу перегляду історії

Рисунок 2.4 – Схема бізнес-процесу генерації питань

2.2 Архітектура програмного забезпечення

Засновуючись на вимогах програмного забезпечення було прийнято рішення використати для реалізації рішення наступний перелік технологій:

а) Серверна частина:

- мова програмування Python;
- середу розробки PyCharm Ultimate Edition;
- фреймворк Django;
- СКБД PostgreSQL.

б) Клієнтська частина

- HTML, CSS;
- фреймворк Bootstrap;

Для реалізації сервісу генерації питань було обрано мову програмування Python, тому що це лаконічна мова, з якою зручно аналізувати та швидко оброблювати великі обсяги даних, що підходить під нашу задачу, так як потрібно для кожного речення формувати звіти з інформацією аналізу декількох NLP моделей.

У якості середи розробки було обрано PyCharm Ultimate Edition, тому що це найбільш зручна та швидка IDE для написання великих за обсягом програм на Python.

Для розроблення веб-застосування було вирішено застосувати Django Framework, тому що ООТВ надаються багато важливих функцій, без яких неможливо було створити веб застосування, таких як робота з базами даних, підтримка веб-інтерфейсу, реалізація захищеного підключення до сервісу та інш.

Для веб інтерфейсу було обрано фреймворк Bootstrap, тому що він містить багато різноманітних шаблонів, які легко можна адаптувати під наше веб застосування, що значно зменшить час розробки.

Усі зазначені вище технології активно підтримується і продовжують удосконалюватися, що забезпечує можливість простого додавання додаткової функціональності у наш сервіс у майбутньому.

Веб-застосування побудовано на основі архітектурного шаблону MVC (Model - View - Controller, Модель - Подання (Вид) – Контролер). Принцип MVC, дозволяє розділити реалізацію логіки, графічний інтерфейс і взаємодію з користувачем. Розглянемо докладніше компоненти, що зображені на рисунку 2.5.

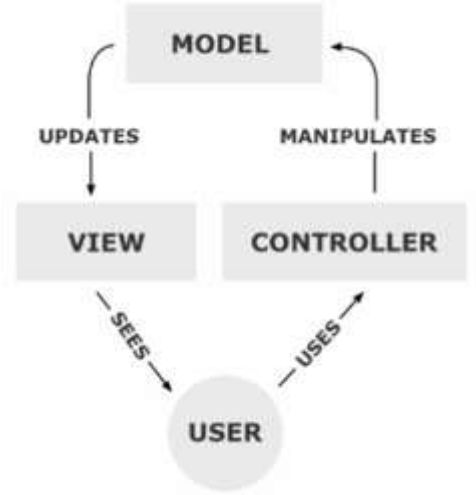


Рисунок 2.5 – Модель MVC

Model – містить бізнес-логіку та DAO, виконує обробку і верифікацію даних, звернення до баз даних.

View - описує зовнішній вигляд програми, графічний інтерфейс, який будуть використовувати користувачі для взаємодії з системою.

Controller - отримує дані від користувача, передає їх моделі, отримує оброблений результат і передає його в представлення.

Щоб забезпечити взаємодію клієнтської та серверної частини, було обрано протокол HTTPS, який реалізовує функцію шифрування з метою підвищення безпеки. Серверна частина веб застосунку має API, що задовольняє вимогам REST архітектури. Для серіалізації даних був обраний формат JSON.

Так як користувач може бути авторизований та неавторизований у системі, то для забезпечення даної функціональності необхідно застосувати програмний модуль ООТВ django.contrib.auth, який надає можливість керувати сесіями користувачів, реалізовувати логіку реєстрації та авторизації. Такими чином виключається можливість несанкціонованого доступу без наявності облікового запису.

Була застосована база даних PostgeSQL для збереження результатів обробки текстів для авторизованих користувачів. Під час проектування бази даних створені наступні таблиці, список та опис наведено у таблиці 2.1.

Таблиця 2.1 – Опис таблиць бази даних

Назва таблиці	Опис таблиці
USERS	Таблиця, що містить у собі логіни та паролі користувачів до системи
TEXTS	Таблиця, що містить у собі тексти, які авторизовані користувачі обробляли раніше
QUESIONS	Таблиця, що містить у собі згенеровані питання до текстів, які авторизовані користувачі обробляли раніше
ANSWERS	Таблиця, що містить у собі варінти відповідей до згенерованих питань, які авторизовані користувачі обробляли раніше

Схема бази даних зображена на рисунку 2.6.

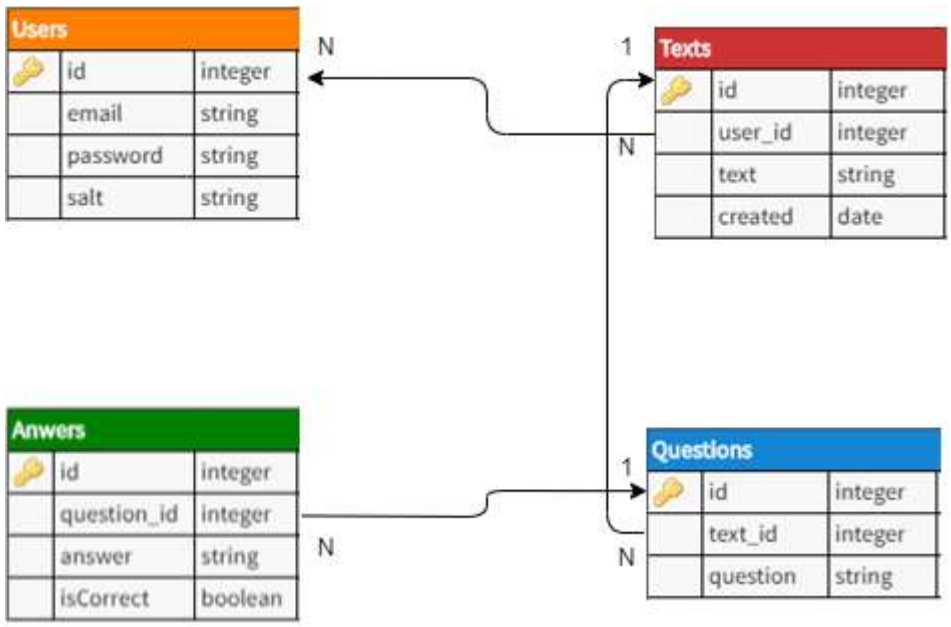


Рисунок 2.6 – Схема бази даних

У таблицях 2.2 - 2.5 представлений детальний опис кожного параметру з усіх таблиць спроектованої бази даних з інформацією щодо заповнення, первинних ключів тощо.

Таблиця 2.2 – Опис таблиці користувачів

Назва	id	email	password	salt
Опис	Ідентифікатор запису	Почта користувача	Пароль користувача	Соль для хешування даних
Тип даних	integer	string	string	string
Обов’язкове	Так	Так	Так	Так
Унікальне	Так	Так	Ні	Так
Первинний ключ	Так	Ні	Ні	Ні
Зовнішній ключ	Ні	Ні	Ні	Ні

Таблиця 2.3 – Опис таблиці текстів

Назва	id	user_id	text	created
Опис	Ідентифікатор запису	Ідентифікатор користувача	Текст	Дата обробки
Тип даних	integer	integer	string	date
Обов’язкове	Так	Так	Так	Так
Унікальне	Так	Ні	Ні	Ні
Первинний ключ	Так	Ні	Ні	Ні
Зовнішній ключ	Ні	Так	Ні	Ні

Таблиця 2.4 – Опис таблиці питань

Назва	id	text_id	question
Опис	Ідентифікатор запису	Ідентифікатор тексту	Питання
Тип даних	integer	integer	string
Обов’язкове	Так	Так	Так
Унікальне	Так	Ні	Ні
Первинний ключ	Так	Ні	Ні
Зовнішній ключ	Ні	Так	Ні

Таблиця 2.5 – Опис таблиці відповідей

Назва	id	question_id	answer	isCorrect
Опис	Ідентифікатор запису	Ідентифікатор питання	Текст відповіді	Чи це правильна відповідь
Тип даних	integer	integer	string	boolean
Обов’язкове	Так	Так	Так	Так
Унікальне	Так	Ні	Ні	Ні
Первинний ключ	Так	Ні	Ні	Ні
Зовнішній ключ	Ні	Так	Ні	Ні

Веб застосування спроектовано з використанням об’єктно-орієнтованого підходу. Діаграму класів можна побачити у додатках до дипломної роботи. Класи додатку було структуровано за допомогою групування їх в окремі модулі на основі виконуваних функцій:

- контролери;
- сервіси;
- класи предметної області;
- репозиторії.

Перелік згрупованих по модулям класів та їх функцій наведено в таблицях 2.6 - 2.9.

Таблиця 2.6 – Опис класів-контролерів

Назва класу	Перелік функції
AuthController	- Створення облікового запису користувача (реєстрація). - Авторизація користувача.
QuestionController	- Обробка введеного користувачем тексту для генерації тестових завдань. - Отримання фільтрів користувача по ключовим словам. - Отримання фільтрів користувача по реченням. - Отримання фільтрів користувача по тегам. - Генерація питань.
HistoryController	- Надання користувачу історії роботи з веб застосуванням.

Таблиця 2.7 – Опис класів-сервісів

Назва класу	Перелік функції
GensimModelService	- Пошук варіантів відповідей для створення тестового питання.

Продовження таблиці 2.7

MorphologicalModelService	- Морфологічний аналіз вхідного речення.
NerModelService	- Пошук іменованих сутностей у вхідному реченні.
TextProcessorService	- Токенезація речень у вхідному тексті. - Пошук та заміна займенників у реченнях.
GenerateQuestionsService	- Генерація питань до тексту. - Формування варіантів відповідей до кожного тестового питання.
AbstractQuestionService	- Генерація питань з заповнюванням пропусків у реченнях.
PersonQuestionService	- Генерація тестів с питальним словом до персон.
DateQuestionService	- Генерація тестів с питальним словом до дат.
GpeQuestionService	- Генерація тестів с питальним словом до локацій.
LocationQuestionService	- Генерація тестів с питальним словом до географічних об’єктів.

Таблиця 2.8 – Опис класів предметної області

Назва класу	Перелік функції
Adjective	- Збереження даних морфологічного аналізу для прикметників.
Adverb	- Збереження даних морфологічного аналізу для прислівників.

Продовження таблиці 2.8

Noun	- Збереження даних морфологічного аналізу для іменників.
Adverb	- Збереження даних морфологічного аналізу для прислівників.
Pronoun	- Збереження даних морфологічного аналізу для займенників.
Verb	- Збереження даних морфологічного аналізу для дієслів.
Word	- Збереження даних морфологічного аналізу для інших слів.
Question	- Збереження даних для створених питань та тестових завдань.
Sentence	- Збереження даних для вхідних речень питань.

Таблиця 2.9 – Опис класів-репозиторіїв

Назва класу	Перелік функції
UserRepository	- Створення запису облікового запису користувача у БД. - Пошук облікового запису користувача у БД.
TextRepository	- Створення запису опрацьованих текстів у БД. - Пошук записів опрацьованих текстів по користувачу у БД.
QuestionRepository	- Створення запису згенерованих питань у БД.

Продовження таблиці 2.9

	- Пошук записів згенерованих питань по тексту у БД.
AnswerRepository	- Створення запису згенеровних відповідей у БД. - Пошук записів згенерованих відповідей по питанням у БД.

2.3 Аналіз безпеки даних

Захист при виконанні HTTP-запитів забезпечує використання протоколу HTTPS, виключаючи можливість перехоплення даних.

Захист особистих даних користувачів забезпечує метод їх зберігання у базі даних:

$$Argon2(SHA1(password), salt),$$

де $Argon2$ – функція формування ключа, $SHA1$ – алгоритм криптографічного хешування, $password$ – стандартний пароль користувача, $salt$ – сіль, специфічна для кожного запису. Такий рівень зберігання паролів забезпечує високу надійність від спроб за короткий час підібрати пароль методами брутфорсингу, з використанням радужних таблиць та інших метод несанкціонованого доступу до даних.

2.4 Висновки по розділу

У даному розділі було проведено змістовний аналіз бізнес процеси розроблюваної системи у вигляді BPMN діаграм. Було визначено архітектуру системи, яка буде використовуватися при розробці сервісу та технології, які будуть застосовуватися.

Також була спроектована модель та структура бази даних для детально описані зв’язки між таблицями.

Було проведено аналіз безпеки даних та встановлено, що спроектована система на має вразливості до несанкціонованому доступу до даних завдяки методам захисту особистої інформації у базі даних та захищеного з'єднання з сервісом.

3 АНАЛІЗ ЯКОСТІ ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Вступ

Якість програмного забезпечення - це сукупність певних характеристик розробленого веб застосування, що відповідають до його можливості задовольняти встановлені та очікувані функціональні вимоги. Для проведення випробувань та визначення відповідності реальної поведінки системи описаним вимогам був розроблений план тестування відповідно до міжнародного стандарту IEEE 892-2008.

3.2 Об’єкти тестування

Система реалізована у вигляді веб-сайту, тому об’єктами тестування є клієнтська та серверна частина.

Клієнтська частина має бути протестована у браузері Google Chrome версії 80.0 або вище.

Серверна частина має бути протестована у середовищі наступними вимогами:

- Linux/Windows операційна система;
- об’єм оперативної пам’яті 1 Гб або вище;
- об’єм дискового простору 10 Гб або вище.

3.3 Функціональність, що підлягає тестуванню

Система має реалізовувати наступні функції

- реєстрація;
- авторизація;
- можливість введення тексту для опрацювання;
- можливість завантаження і конвертації документів формату txt, doc та pdf;

- можливість вибирати ключові слова для формування питань;
- можливість вибирати речення для формування питань;
- можливість вибирати теги для формування питань;
- можливість генерації питань по тексту;
- можливість експортувати результат у форматі txt, doc та pdf документів;
- перегляд історії опрацьованих текстів та згенерованих питань;
- вихід з системи.

3.4 Функціональність, що не підлягає тестуванню

Не підлягає тестуванню наступна функціональність

- зручність у використанні;
- дизайн.

3.5 Методологія проведення тестування

Тестувальник має провести тест кожного прецеденту з use-case діаграми та занотувати до звіту усі отримані результати. Якщо протягом тестування були знайдені баги або дефекти, звіт з усією інформацію передається розробнику програмного забезпечення для виправлення.

3.6 Критерії проходження/провалу тестування

Вся система повинна функціонувати так, як описано в функціональних вимогах до реалізації сервісу. Усі unit-тести мають бути пройдені. Не має бути відкритих та не розрішених дефектів або багів до системи.

3.7 Критерії призупинення тестування

Процес тестування призупиняється, якщо був знайдений дефект типу Blocker, що впливає на функціональну працездатність усієї системи.

Тестування має бути продовжене, коли розробник вирішить цю проблему та наддасть нову версію програмного забезпечення.

3.8 Вимоги до тестового середовища

Спеціальних вимог до тестового середовища немає.

3.9 Відповідальність

Відповідальною людиною за графік тестування, своєчасне повідомлення про знайдені дефекти є тестувальник застосунку.

3.10 Розклад

Процес тестування починається, коли усі основні функції сервісу реалізовані та не пізніше, чим за 3 тижні до дати введення продукту в експлуатацію.

3.11 Ризики та непередбачувані ситуації

Тестування може бути відкладене або не повністю завершене у випадку форс-мажорних обставин, які були обговорені перед початком виконування реалізації сервісу. У такому випадку тестувальник не несе відповідальності за несвоєчасне закінчене тестування.

3.12 Схвалення

Процес тестування набуває схваленого статусу, якщо усі функціональні вимоги були перевірені та відсутні критичні дефекти, які роблять неможливим нормальне функціонування програмного забезпечення. Також вся логіка програми має бути покрита unit тестами та усі вони мають проходити.

3.13 Опис тестових прецедентів

Перелік основних тестових прецедентів, для яких буде проводиться тестування, наведено у таблицях 3.1 -

Таблиця 3.1 – Тестовий прецедент “Реєстрація у системі”

Назва	Реєстрація у системі
Передумови	Користувач незареєстрований у системі.
Кроки виконання	- Перейти до основної сторінки веб застосування. - Натиснути кнопку “Sign up”. - У формі авторизації, ввести емайл, нік та пароль до нового облікового запису. - Натиснути кнопку “Sign up”.
Очікуваний результат	Повідомлення про успішну реєстрацію. Користувач перенаправляється на головну сторінку вже авторизованим.
Отриманий результат	Повідомлення про успішну реєстрацію. Користувач перенаправляється на головну сторінку вже авторизованим.

Таблиця 3.2 – Тестовий прецедент “Авторизація у системі”

Назва	Авторизація у системі
Передумови	Користувач неавторизований у системі.
Кроки виконання	- Перейти до основної сторінки веб застосування. - Натиснути кнопку “Log in”. - У формі авторизації, ввести нік та пароль до існуючого облікового запису. - Натиснути кнопку “Log in”.

Продовження таблиці 3.2

Очікуваний результат	Повідомлення про успішну авторизацію. Користувач перенаправляється на головну сторінку вже авторизованим.
Отриманий результат	Повідомлення про успішну авторизацію. Користувач перенаправляється на головну сторінку вже авторизованим.

Таблиця 3.3 – Тестовий прецедент “Завантаження тексту у форматі doc”

Назва	Завантаження тексту у форматі doc
Передумови	
Кроки виконання	- Перейти до сторінки “Create” веб застосування. - Додати текст до форми Натиснути кнопку “Choose file”. - Обрати файл doc формату та натиснути кнопку “Open”.
Очікуваний результат	Текст з файлу конвертується до форми.
Отриманий результат	Текст з файлу конвертується до форми.

Таблиця 3.4 – Тестовий прецедент “Завантаження тексту у форматі txt”

Назва	Завантаження тексту у форматі txt
Передумови	
Кроки виконання	- Перейти до сторінки “Create” веб застосування. - Додати текст до форми Натиснути кнопку “Choose file”. - Обрати файл txt формату та натиснути кнопку “Open”.

Продовження таблиці 3.4

Очікуваний результат	Текст з файлу конвертується до форми.
Отриманий результат	Текст з файлу конвертується до форми.

Таблиця 3.5 – Тестовий прецедент “Завантаження тексту у форматі pdf”

Назва	Завантаження тексту у форматі pdf
Передумови	
Кроки виконання	- Перейти до сторінки “Create” веб застосування. - Додати текст до форми Натиснути кнопку “Choose file”. - Обрати файл pdf формату та натиснути кнопку “Open”.
Очікуваний результат	Текст з файлу конвертується до форми.
Отриманий результат	Текст з файлу конвертується до форми.

Таблиця 3.5 – Тестовий прецедент “Встановлення фільтрів за реченнями”

Назва	Встановлення фільтрів за реченнями
Передумови	Наявність сконвертованого тексту.
Кроки виконання	Для вибору речень для генерації питань необхідно застосовувати checkbox біля кожного речення.
Очікуваний результат	Фільтри за реченнями встановлені.
Отриманий результат	Фільтри за реченнями встановлені.

Таблиця 3.6 – Тестовий прецедент “Встановлення фільтрів за ключовими словами”

Назва	Встановлення фільтрів за ключовими словами
Передумови	Наявність конвертованого тексту.

Продовження таблиці 3.6

Кроки виконання	Для вибору ключових слів для генерації питань у реченнях необхідно натискати на кнопку конкретної іменованої сутності у реченні, щоб вона придбала насичений відтінок.
Очікуваний результат	Фільтри за ключовими словами встановлені.
Отриманий результат	Фільтри за ключовими словами встановлені.

Таблиця 3.7 – Тестовий прецедент “Встановлення фільтрів за тегами”

Назва	Встановлення фільтрів за тегами
Передумови	Наявність конвертованого тексту.
Кроки виконання	Для вибору тегів для генерації питань у тексті необхідно натискати на кнопки конкретного тега, які розташовані вище згенерованих речень, щоб вони придбали насичений відтінок.
Очікуваний результат	Фільтри за тегами встановлені.
Отриманий результат	Фільтри за тегами встановлені.

Таблиця 3.8 – Тестовий прецедент “Експорт згенерованих питань у doc файл”

Назва	Експорт згенерованих питань у doc файл
Передумови	Наявність конвертованого питань.
Кроки виконання	Натиснути кнопку “Download doc”.
Очікуваний результат	Файл формату doc був завантажений.
Отриманий результат	Файл формату doc був завантажений.

Таблиця 3.9 – Тестовий прецедент “Експорт згенерованих питань у txt файл”

Назва	Експорт згенерованих питань у txt файл

Продовження таблиці 3.9

Передумови	Наявність згенерованих питань.
Кроки виконання	Натиснути кнопку “Download txt”.
Очікуваний результат	Файл формату txt був завантажений.
Отриманий результат	Файл формату txt був завантажений.

Таблиця 3.10 – Тестовий прецедент “Експорт згенерованих питань у pdf файл”

Назва	Експорт згенерованих питань у pdf файл
Передумови	Наявність згенерованих питань.
Кроки виконання	Натиснути кнопку “Download pdf ”.
Очікуваний результат	Файл формату pdf був завантажений.
Отриманий результат	Файл формату pdf був завантажений.

3.14 Висновки до розділу

У даному розділі було розроблено змістовний план тестування реалізованого програмного забезпечення. На його основі буде можливість провести повне тестування сервісу та виявити усі потенційні дефекти, які можуть призвести до повного нефункціонування реалізованого сервісу, що може спричинити небажану втрату користувачів та отримання негативного досвіду роботи з системою.

4 ВПРОВАДЖЕННЯ ТА СУПРОВІД ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Розгортання програмного забезпечення

Для того, щоб розгорнути веб застосування, необхідна наявність наступних компонентів:

- операційна система Windows/Linux;
- Python версії 3.7 або вище;
- СКБД PostgreSQL версії 12.0 або вище;
- GitBash версії 2.21.0 або вище.

Програмний продукт надається у форматі GitHub репозиторію. Необхідно виконати наступні дії для розгортання веб сервісу:

- склонувати репозиторій проекту, використовуючи команду *git clone https://github.com/SanekTNT/Diploma.git*;
- обрати каталог для віртуальної середи проекту та виконати команду *virtualenv project*;
- у створеному каталозі за шляхом *project\Scripts* потрібно активувати віртуальну середу виконавши *activate.bat* файл;
- підключити Django фреймворк командую *pip install django* у каталозі з встановленою віртуальною середою;
- виконати команду *django-admin startproject newProject* для створення нового Django проекту під наш веб застосунок;
- у створеному каталозі для нового Django проекту виконати команду *python manage.py runserver* для запуска серверу з веб застосунком;
- сервіс готовий до роботи за шляхом *http://127.0.0.1:8000/;*

Варто відзначити, що не потрібно скачувати та встановлювати NLP моделі для обробки речень, вони автоматично підтягнуться протягом першого запуску програми.

4.2 Робота з програмним забезпеченням

Детальна інструкція з експлуатації веб застосунку та приклади використання наведена у документі Г «Керівництво користувача».

ПЕРЕЛІК ПОСИЛАНЬ

1) Обработка естественного языка [Електронний ресурс] // Wikipedia. – 2020. – Режим доступу до ресурсу: https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%BA%D0%B0_%D0%B5%D1%81%D1%82%D0%B5%D1%81%D1%82%D0%B2%D0%B5%D0%BD%D0%BD%D0%BE%D0%B3%D0%BE_%D1%8F%D0%B7%D1%8B%D0%BA%D0%B0

2) NLP. Основы. Техники. Саморазвитие. Часть 2: NER [Електронний ресурс] // Хабр. – 2020. – Режим доступу до ресурсу: https://habr.com/ru/company/abbyy/blog/449514/

3) DeepPavlov library documentation [Електронний ресурс] // Python docs. – 2020. – Режим доступу до ресурсу: http://docs.deeppavlov.ai/en/master/

4) Обработка естественного языка на Python [Електронний ресурс] // Proglib. – 2020. – Режим доступу до ресурсу: https://proglib.io/p/fun-nlp

5) Word2vec в картинках с примерами [Електронний ресурс] // Хабр. – 2020. – Режим доступу до ресурсу: https://habr.com/ru/post/446530/

6) Gensim documentation - [Електронний ресурс] // Gensim. – 2020. – Режим доступу до ресурсу: https://radimrehurek.com/gensim/auto_examples/index.html

7) Полный синтаксический разбор предложения русского языка [Електронний ресурс] // Хабр. – 2020. – Режим доступу до ресурсу: https://habr.com/ru/post/464959/

8) Model-View-Controller [Електронний ресурс] // Wikipedia. – 2020. – Режим доступу до ресурсу: https://ru.wikipedia.org/wiki/Model-View-Controller

9) Software test documentation [Електронний ресурс] // Wikipedia. – 2020. – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Software_test_documentation

10) Bootstrap documentation [Електронний ресурс] //Bootstrap. – 2020. – Режим доступу до ресурсу: https://getbootstrap.com/docs/4.5/getting-started/introduction/

ВИСНОВКИ

У даній роботі була виконана розробка веб застосування для автоматичного процесу генерації тестових питань за текстовими даними.

Результатом даної роботи є створення та аналіз ефективності використання алгоритму у сервісу для вирішення задачі генерації питань по тексту без участі людини.

У процесі написання даної дипломної роботи був виконаний детальний аналіз предметної області, розібрані основні підходи до вирішення проблеми.

Були досліджені основні аналоги, які на сьогоднішній день представлені на ринку, виділені їх основні переваги та недоліки. Було встановлено, що жодна з існуючих реалізацій не орієнтована на обробку текстів українською мовою і генерації тестових питань з них, тому в рамках даної дипломної роботи було розроблено веб сервіс, що вирішує цю проблему.

Проведено аналіз вимог до програмного забезпечення у вигляді розробки моделі функціональних та нефункціональних вимог. Були визначені залежності між функціональними вимогами та варіантами використання, а також побудовані діаграми матриці трасування та use-case діаграма використання сервісу.

На основі моделі варіантів використання було виконане моделювання програмного забезпечення з точки зору архітектурної реалізації. Створені бізнес-процеси були описані у вигляді діаграм активності та BPMN-діаграм.

Були описані основні технології, які застосовувалися в процесі імплементації даного сервісу та спроектована база даних та описані поля таблиць та зв’язки між ними.

Був виконаний аналіз безпеки даних створеного програмного забезпечення. Під час аналізу було отримано задовільні результати стосовно безпеки зберігання даних у базі даних та обміну даними, який відбувається через захищене підключення.

Для забезпечення високого рівню якості створеного веб застосунка був розроблений план та виконане тестування, результати задовільні.

Була розроблена інструкція користування програмним забезпечення з прикладами використання та сформовано звіт по процесу розробки даного програмного забезпечення.

Факультет інформатики та обчислювальної техніки

Кафедра автоматизованих систем обробки інформації і управління

“ЗАТВЕРДЖЕНО”

В.о. завідувача кафедри

_____ Олександр ПАВЛОВ

“___” _____ 2020 р.

ВЕБ ЗАСТОСУВАННЯ ДЛЯ ГЕНЕРАЦІЇ ПИТАНЬ ЗА ТЕКСТОВИМИ ДАНИМИ

Технічне завдання

КПІ.ІП-6319.045440.02.91

“ПОГОДЖЕНО”

Керівник проєкту:

_____ Іванова Л.М.

Нормоконтроль:

_____ К.І. Ліщук

Виконавець:

_____ Міщенко О.О.

Київ – 2020 року

ЗМІСТ

1 НАЙМЕНУВАННЯ ТА ГАЛУЗЬ ЗАСТОСУВАННЯ

Назва розробки: Веб застосування для генерації питань за текстовими даними

Галузь застосування:

Наведене технічне завдання поширюється на розробку веб застосування для аналізу та обробки тексту, на базі якого будуть генеруватись питання та тести для контролю знань. Користувачами можуть бути викладачі, що мають перевірити засвоєння наукового матеріалу методом тестових питань. Застосування також може бути використане у будь-якій сфері, де потрібно відстежувати якість освоєння наукового матеріалу.

2 ПІДСТАВА ДЛЯ РОЗРОБКИ

Підставою для розробки веб застосування для генерації питань за текстовими даними є завдання на дипломне проектування, затверджене кафедрою автоматизованих систем обробки інформації і управління Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського» (КПІ ім.Ігоря Сікорського).

3 ПРИЗНАЧЕННЯ РОЗРОБКИ

Розробка призначена для автоматизації процесу аналізу тексту на предмет ключових слів, за якими можуть будти створені тестові завдання. Отримані дані можуть бути використані користувачем для формування звіту створених по тексту питань.

Метою створення розробки є зменшення витрат людського часу на аналіз та обробку текстів з підручників, статей та лекцій для створення тестових завдань.

4 ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Вимоги до функціональних характеристик

4.1.1 Програмне забезпечення повинно забезпечувати виконання наступних основних функції для користувача:

- реєстрація;
- авторизація;
- можливість введення тексту для опрацювання;
- можливість завантаження і конвертації документів формату txt, doc та pdf;
- можливість вибирати ключові слова для формування питань;
- можливість вибирати речення для формування питань;
- можливість вибирати теги для формування питань;
- можливість генерації питань по тексту;
- можливість експортувати результат у форматі txt, doc та pdf документів;
- перегляд історії опрацьованих текстів та згенерованих питань;
- вихід з системи.

4.1.2 Вимоги до організації вхідних даних

- текст;
- текстовий файл у форматі txt, doc та pdf.

4.1.3 Розробку виконати на будь-якій платформі (Windows/Linux)

4.1.4 Додаткові вимоги

Не передбачені.

4.2 Вимоги до надійності

4.2.1 Передбачити контроль введення інформації.

4.2.2 Передбачити захист від некоректних дій користувача.

4.3 Умови експлуатації

4.3.1 Умови експлуатації згідно СанПін 2.2.2.542 – 96.

Не висуваються.

4.3.2 Обслуговування та обслуговуючий персонал.

Не висуваються.

4.4 Вимоги до складу і параметрів технічних засобів

4.4.1 Програмне забезпечення повинно функціонувати на серверах з операційною системою Windows/Linux.

4.4.2 Мінімальна конфігурація технічних засобів:

4.4.2.1 Тип процесору

Будь-який процесор що підтримує ОС Windows/Linux та Python.

4.4.2.2 Об‘єм ОЗП

1 Гб або більше

4.5 Вимоги до інформаційної та програмної сумісності

- програмне забезпечення повинно працювати під управлінням операційної системи Windows/Linux;
- вхідні дані повинні бути представлені в наступному форматі: HTTP-запит до серверу;

- вихідні дані повинні бути представлені в наступному форматі: HTTP-відповідь з серверу.

4.6 Вимоги до маркування та пакування

Вимоги до маркування та пакування не висуваються.

4.7 Вимоги до транспортування та зберігання

Вимоги до транспортування та зберігання не висуваються.

4.8 Спеціальні вимоги

Згенерувати установочну версію програмного забезпечення.

5 ВИМОГИ ДО ПРОГРАМНОЇ ДОКУМЕНТАЦІЇ

5.1 Попередній склад програмної документації

а) Супроводжувальна документація:

1) пояснювальна записка;

2) керівництво користувача;

3) програма та методика тестування.

б) Довідникова документація:

1) програмні модулі, котрі розробляються, повинні бути задокументовані, тобто тексти програм повинні містити всі необхідні коментарі;

2) програмне забезпечення повинно мати вбудовану довідку.

в) Графічна документація:

1) схема структурна варіантів використання.

2) креслення вигляду екранних форм.

3) схема структурна класів програмного забезпечення.

6 СТАДІЇ І ЕТАПИ РОЗРОБКИ

№	Назва етапу	Строк	Звітність
1.	Вивчення літератури за тематикою проекту	22.04.2020	
2.	Розробка технічного завдання	24.04.2020	Технічне завдання
3.	Аналіз вимог та уточнення специфікацій	27.04.2020	Специфікації програмного забезпечення
4.	Проектування структури програмного забезпечення, проектування компонентів	04.05.2020	Схема структурна програмного забезпечення та специфікація компонентів (діаграма класів, схема алгоритму …)
5.	Програмна реалізація програмного забезпечення	07.05.2020	Тексти програмного забезпечення
6.	Тестування програмного забезпечення	08.05.2020	Тести, результати тестування
7.	Розробка матеріалів текстової частини проекту	12.05.2020	Пояснювальна записка.
8.	Розробка матеріалів графічної частини проекту	13.05.2020	Графічний матеріал проекту
9.	Оформлення технічної документації проекту	15.05.2020	Технічна документація

7 ПОРЯДОК КОНТРОЛЮ ТА ПРИЙМАННЯ

7.1 Види випробувань

Тестування описано в пояснювальній записці «КПІ.ІП-6319.045440.02.81», розділі 3 «Аналіз та тестування програмного забезпечення».

Факультет інформатики та обчислювальної техніки

Кафедра автоматизованих систем обробки інформації і управління

“ЗАТВЕРДЖЕНО”

В.о. завідувача кафедри

_____ Олександр ПАВЛОВ

“___” _____ 2020 р.

ВЕБ-ЗАСТОСУВАННЯ ДЛЯ ГЕНЕРАЦІЇ ПИТАНЬ ЗА ТЕКСТОВИМИ ДАНИМИ

Опис програми

КПІ.ІП-6319.045440.03.13

“ПОГОДЖЕНО”

Керівник проєкту:

_____ Л.М. Іванова

Нормоконтроль:

_____ К.І. Ліщук

Виконавець:

_____О.О. Міщенко

Київ – 2020 року

Тексти програмного коду

Веб застосування для генерації питань за текстовими даними

(Найменування програми (документа))

DVD-R

(Вид носія даних)

48 арк, 186 Кб

(Обсяг програми (документа) , арк.,) Кб)

Київ – 2020

```
from algorithm.dto.word.Adjective import Adjective
from algorithm.dto.word.Adverb import Adverb
from algorithm.dto.word.Determinant import Determinant
from algorithm.dto.word.Noun import Noun
from algorithm.dto.word.Pronoun import Pronoun
from algorithm.dto.word.Verb import Verb
from algorithm.dto.word.Word import Word
from algorithm.service.enum.PartOfSpeech import PartOfSpeech
from algorithm.service.enum.WordTag import WordTag


def __create_noun(word, tag, morpho, number_in_sentence):
    return Noun(word, WordTag[tag], morpho[2], PartOfSpeech[morpho[3]],
number_in_sentence, __extract_animacy(morpho),
                __extract_case(morpho),
                __extract_gender(morpho), __extract_number(morpho))


def __create_verb(word, tag, morpho, number_in_sentence):
    return Verb(word, WordTag[tag], morpho[2], PartOfSpeech[morpho[3]],
number_in_sentence, __extract_aspect(morpho),
                __extract_gender(morpho),
                __extract_mood(morpho), __extract_number(morpho),
__extract_tense(morpho),
                __extract_verb_form(morpho), __extract_voice(morpho))


def __create_adjective(word, tag, morpho, number_in_sentence):
```

```
    return Adjective(word, WordTag[tag], morpho[2], PartOfSpeech[morpho[3]],
number_in_sentence, __extract_case(morpho),
                     __extract_degree(morpho),
                     __extract_gender(morpho), __extract_number(morpho))


def __create_pronoun(word, tag, morpho, number_in_sentence):
    return Pronoun(word, WordTag[tag], morpho[2], PartOfSpeech[morpho[3]],
number_in_sentence, __extract_case(morpho),
                   __extract_gender(morpho),
                   __extract_number(morpho), __extract_person(morpho))


def __create_adverb(word, tag, morpho, number_in_sentence):
    return Adverb(word, WordTag[tag], morpho[2], PartOfSpeech[morpho[3]],
number_in_sentence, __extract_degree(morpho))


def __create_determinant(word, tag, morpho, number_in_sentence):
    return Determinant(word, WordTag[tag], morpho[2], PartOfSpeech[morpho[3]],
number_in_sentence,
                       __extract_case(morpho),
                       __extract_number(morpho))


def __create_word(word, tag, morpho, number_in_sentence):
    return Word(word, WordTag[tag], morpho[2], PartOfSpeech[morpho[3]],
number_in_sentence)


def __extract_animacy(morpho):
```

```
    animacy = 'Animacy='
    for data in morpho:
        if animacy in data:
            return data.replace(animacy, '')
    return empty_string


def __extract_case(morpho):
    case = 'Case='
    for data in morpho:
        if case in data:
            return data.replace(case, '')
    return empty_string


def __extract_gender(morpho):
    gender = 'Gender='
    for data in morpho:
        if gender in data:
            return data.replace(gender, '')
    return empty_string


def __extract_number(morpho):
    number = 'Number='
    for data in morpho:
        if number in data:
            return data.replace(number, '')
    return empty_string
```

```
def __extract_aspect(morpho):
    aspect = 'Aspect='
    for data in morpho:
        if aspect in data:
            return data.replace(aspect, '')
    return empty_string


def __extract_mood(morpho):
    mood = 'Mood='
    for data in morpho:
        if mood in data:
            return data.replace(mood, '')
    return empty_string


def __extract_tense(morpho):
    tense = 'Tense='
    for data in morpho:
        if tense in data:
            return data.replace(tense, '')
    return empty_string


def __extract_verb_form(morpho):
    veb_form = 'VerbForm='
    for data in morpho:
        if veb_form in data:
            return data.replace(veb_form, '')
```

```
    return empty_string


def __extract_voice(morpho):
    voice = 'Voice='
    for data in morpho:
        if voice in data:
            return data.replace(voice, '')
    return empty_string


def __extract_degree(morpho):
    degree = 'Degree='
    for data in morpho:
        if degree in data:
            return data.replace(degree, '')
    return empty_string


def __extract_person(morpho):
    person = 'Person='
    for data in morpho:
        if person in data:
            return data.replace(person, '')
    return empty_string


word_creating_methods_by_type = {
    PartOfSpeech.NOUN: __create_noun,
    PartOfSpeech.PROPN: __create_noun,
```

```
    PartOfSpeech.VERB: __create_verb,
    PartOfSpeech.AUX: __create_verb,
    PartOfSpeech.ADJ: __create_adjective,
    PartOfSpeech.PRON: __create_pronoun,
    PartOfSpeech.ADV: __create_adverb,
    PartOfSpeech.DET: __create_determinant,
    PartOfSpeech.CCONJ: __create_word,
    PartOfSpeech.SCONJ: __create_word,
    PartOfSpeech.ADP: __create_word,
    PartOfSpeech.PUNCT: __create_word,
    PartOfSpeech.NUM: __create_word,
    PartOfSpeech.SYM: __create_word,
    PartOfSpeech.PART: __create_word,
    PartOfSpeech.X: __create_word
}


def convert_to_word(word, tag, morpho, number_in_sentence):
    return word_creating_methods_by_type[PartOfSpeech[morpho[3]]](word.lower(),
tag, morpho, number_in_sentence)

import logging

from algorithm.dto.Sentence import Sentence
from algorithm.methods.createword import convert_to_word


def extract(ner_pairs, morphological_pairs):
    result = []
    for (ner_pair, morphological_pair) in zip(ner_pairs, morphological_pairs):
```

```
        words = __split_sentence_to_words(ner_pair[0], ner_pair[1],
morphological_pair[1])
        if words:
            result.append(Sentence(words))
    return result


def __split_sentence_to_words(sentence, word_tags, word_morphos):
    try:
        result = []
        for i in range(0, len(sentence)):
            result.append(convert_to_word(sentence[i], word_tags[i], word_morphos[i],
i))
        return result
    except IndexError:
        __log_splitting_sentence_error(sentence, word_tags, word_morphos)
    return []


def __log_splitting_sentence_error(sentence, word_tags, word_morphos):
    logging.error("Error converting sentence: " + ' '.join(sentence))
    logging.error("Ner model output: " + ' '.join(word_tags))
    logging.error("Morphological model output: " + '; '.join(' '.join(word) for word in
word_morphos))

from algorithm.service.enum.WordTag import WordTag


def filter_sentences_by_tags(sentences):
    result = {}
```

```
    for sentence in sentences:
        for word in list(filter(lambda w: w.tag != WordTag.O, sentence.words)):
            if word.base_tag not in result:
                result[word.base_tag] = []
            if sentence not in result[word.base_tag]:
                result[word.base_tag].append(sentence)
    return result


from enum import Enum



class PartOfSpeech(Enum):
    NOUN = 'NOUN'
    PROPN = 'PROPN'
    VERB = 'VERB'
    AUX = 'AUX'
    ADJ = 'ADJ'
    PRON = 'PRON'
    ADV = 'ADV'
    DET = 'DET'
    CCONJ = 'CCONJ'
    SCONJ = 'SCONJ'
    ADP = 'ADP'
    PUNCT = 'PUNCT'
    NUM = 'NUM'
    SYM = 'SYM'
    PART = 'PART'
    X = 'X'


from enum import Enum
```

```
class WordTag(Enum):
    PERSON = 'PERSON'
    B_PERSON = 'B_PERSON'
    I_PERSON = 'I_PERSON'
    NORP = 'NORP'
    B_NORP = 'B_NORP'
    I_NORP = 'I_NORP'
    FAC = 'FAC'
    B_FAC = 'B_FAC'
    I_FAC = 'I_FAC'
    ORG = 'ORG'
    B_ORG = 'B_ORG'
    I_ORG = 'I_ORG'
    GPE = 'GPE'
    B_GPE = 'B_GPE'
    I_GPE = 'I_GPE'
    LOC = 'LOC'
    B_LOC = 'B_LOC'
    I_LOC = 'I_LOC'
    PRODUCT = 'PRODUCT'
    B_PRODUCT = 'B_PRODUCT'
    I_PRODUCT = 'I_PRODUCT'
    EVENT = 'EVENT'
    B_EVENT = 'B_EVENT'
    I_EVENT = 'I_EVENT'
    WORK_OF_ART = 'WORK_OF_ART'
    B_WORK_OF_ART = 'B_WORK_OF_ART'
    I_WORK_OF_ART = 'I_WORK_OF_ART'
```

```
LAW = 'LAW'
B_LAW = 'B_LAW'
I_LAW = 'I_LAW'
LANGUAGE = 'LANGUAGE'
B_LANGUAGE = 'B_LANGUAGE'
I_LANGUAGE = 'I_LANGUAGE'
DATE = 'DATE'
B_DATE = 'B_DATE'
I_DATE = 'I_DATE'
TIME = 'TIME'
B_TIME = 'B_TIME'
I_TIME = 'I_TIME'
PERCENT = 'PERCENT'
B_PERCENT = 'B_PERCENT'
I_PERCENT = 'I_PERCENT'
MONEY = 'MONEY'
B_MONEY = 'B_MONEY'
I_MONEY = 'I_MONEY'
QUANTITY = 'QUANTITY'
B_QUANTITY = 'B_QUANTITY'
I_QUANTITY = 'I_QUANTITY'
ORDINAL = 'ORDINAL'
B_ORDINAL = 'B_ORDINAL'
I_ORDINAL = 'I_ORDINAL'
CARDINAL = 'CARDINAL'
B_CARDINAL = 'B_CARDINAL'
I_CARDINAL = 'I_CARDINAL'
O = 'O'


import logging
```

```
import re

from algorithm.service.enum.PartOfSpeech import PartOfSpeech
from algorithm.service.enum.WordTag import WordTag
from loader.gensimmodels import load_gensim_model, load_gensim_number_model


class GensimModelService:

    def __init__(self):
        self.__gensim_model = load_gensim_model()
        self.__gensim_number_model = load_gensim_number_model()

    def analyze(self, sentences):
        result = {}
        for sentence in sentences:
            for word in sentence.words:
                if word.base_tag != WordTag.O:
                    if re.fullmatch(r'^[0-9MmDdCcLlXxVvIi]*$', word.value):
                        similar_numbers = self.__get_similar_numbers(word)
                        logging.info(similar_numbers)
                        if word.lemma not in result:
                            result[word.lemma] =
self.__extract_similar_numbers(similar_numbers)
                    else:
                        similar_words = self.__get_similar_words(word)
                        logging.info(similar_words)
                        if word.lemma not in result:
                            result[word.lemma] = self.__extract_similar_words(similar_words)
                    logging.info('')
```

```
        return result

    def __get_similar_numbers(self, word):
        result = []
        try:
            logging.info(word.value)
            result.extend(self.__gensim_number_model.similar_by_word(word.value))
        except KeyError:
            logging.error('Can not find similar numbers for: ' + str(word))
        return result

    def __extract_similar_numbers(self, similar_numbers):
        result = []
        for pair in similar_numbers:
            result.append(pair[0])
        return result

    def __get_similar_words(self, word):
        result = []
        try:
            string = word.lemma + '_' + (
                word.part_of_speech.value if word.part_of_speech !=
PartOfSpeech.PROPN else PartOfSpeech.NOUN.value)
            logging.info(string)
            result.extend(self.__gensim_model.most_similar(string))
        except KeyError:
            logging.error('Can not find similar words for: ' + str(word))
        return result

    def __extract_similar_words(self, similar_words):
```

```
        result = []
        for pair in similar_words:
            result.append(pair[0])
        return result

import re

from loader.deeppavlovmodels import load_morphological_model


class MorphologicalModelService:

    def __init__(self):
        self.__morphological_model = load_morphological_model()

    def analyze(self, sentences):
        result = []
        morphological_sentences = self.__morphological_model(sentences)
        for i, morph_sentence in enumerate(morphological_sentences):
            result.append((sentences[i],
self.__format_morphological_result(morph_sentence)))
        return result

    @staticmethod
    def __format_morphological_result(sentence):
        result = []
        words = sentence.split('\n')
        for word in words:
            if word != '':
                word = re.sub(r'[\t|]', ' ', word)
```

```
            result.append(word.split(' '))
        return result

from loader.deeppavlovmodels import load_ner_model


class NerModelService:
    __dash = '-'
    __underscore = '_'

    def __init__(self):
        self.__ner_model = load_ner_model()

    def analyze(self, sentences):
        result = []
        ner_result = self.__ner_model(sentences)
        for i, sentence in enumerate(ner_result[0]):
            tags = self.__replace_dash_by_underscore(ner_result[1][i])
            result.append((sentence, tags))
        return result

    def __replace_dash_by_underscore(self, tags):
        return list(map(lambda tag: tag.replace(self.__dash, self.__underscore), tags))

from algorithm.dto.Question import Question
from algorithm.service.enum.PartOfSpeech import PartOfSpeech
from algorithm.service.enum.WordTag import WordTag
from algorithm.service.question.AbstractQuetionService import
AbstractQuestionService
```

```
class DateQuestionService(AbstractQuestionService):

    def __init__(self, base_tag, b_tag, i_tag):
        super().__init__(base_tag, b_tag, i_tag)

    def _generate_special_questions(self, sentence):
        question_part, verbs, dates = self.__find_part_for_question(
            self._split_sentence_by_punctuation(sentence.words), sentence)
        if question_part:
            question_part = self._extract_remaining_words(sentence.words, (verbs,
dates))
            sentence.answer_questions.append(
                Question(self._convert_to_special_question_string_form(question_part,
verbs,

self._extract_remaining_words(sentence.words, (
                                                    question_part, verbs, dates))),
                         self._extract_special_question_answers(dates), sentence,
self._base_tag))

    def __find_part_for_question(self, sentence_parts, sentence):
        for part in sentence_parts:
            dates = self._extract_words_by_base_tag(part, self._base_tag)
            if dates[0].number_in_sentence != 0:
                possible_adp = sentence.words[dates[0].number_in_sentence - 1]
                if possible_adp.tag == WordTag.O and possible_adp.part_of_speech ==
PartOfSpeech.ADP:
                    dates.insert(0, possible_adp)
```

```
            verbs = self._extract_words_by_parts_of_speech(part, (PartOfSpeech.AUX,
PartOfSpeech.VERB))
            if dates and verbs:
                sentence_parts.remove(part)
                return part, verbs, dates
        return [], [], []

    def _convert_to_special_question_string_form(self, question_words, verbs,
other_words):
        result = self._when + ' ' + ' '.join(word.value for word in verbs) + ' ' + ' '.join(
            word.value for word in question_words) + ' '.join(word.value for word in
other_words)
        return self._remove_redundant_punctuation_and_add_question_mark(result)

from algorithm.service.question.AbstractQuetionService import
AbstractQuestionService


class GpeQuestionService(AbstractQuestionService):

    def __init__(self, base_tag, b_tag, i_tag):
        super().__init__(base_tag, b_tag, i_tag)

from algorithm.service.question.AbstractQuetionService import
AbstractQuestionService


class LocationQuestionService(AbstractQuestionService):

    def __init__(self, base_tag, b_tag, i_tag):
```

```
        super().__init__(base_tag, b_tag, i_tag)

from algorithm.dto.Question import Question
from algorithm.dto.word.Noun import Noun
from algorithm.service.enum.PartOfSpeech import PartOfSpeech
from algorithm.service.question.AbstractQuetionService import
AbstractQuestionService


class PersonQuestionService(AbstractQuestionService):

    def __init__(self, base_tag, b_tag, i_tag):
        super().__init__(base_tag, b_tag, i_tag)


    def _generate_special_questions(self, sentence):
        question_part, verbs, nom_persons = self.__find_part_for_question(
            self._split_sentence_by_punctuation(sentence.words))
        if question_part:
            question_part = self._extract_remaining_words(sentence.words, (verbs,
nom_persons))
            sentence.answer_questions.append(
                Question(self._convert_to_special_question_string_form(question_part,
verbs,

self._extract_remaining_words(sentence.words, (
                                                    question_part, verbs, nom_persons))),
                        self._extract_special_question_answers(nom_persons), sentence,
self._base_tag))
```

```
    def __find_part_for_question(self, sentence_parts):
        for part in sentence_parts:
            persons = self.__extract_person_for_special_question_word(part)
            verbs = self._extract_words_by_parts_of_speech(part, (PartOfSpeech.AUX,
PartOfSpeech.VERB))
            if persons and verbs:
                sentence_parts.remove(part)
                return part, verbs, persons
        return [], [], []

    def __extract_person_for_special_question_word(self, words):
        persons = self._extract_words_by_base_tag(words, self._base_tag)
        if self._is_exists_noun_words_by_case(persons, 'Nom'):
            return persons
        return []

    @staticmethod
    def _is_exists_noun_words_by_case(words, case):
        result = list(filter(lambda w: isinstance(w, Noun), words))
        return list(filter(lambda w: w.case == case, result))

    def _convert_to_special_question_string_form(self, question_words, verbs,
other_words):
        result = self._who + ' ' + ' '.join(word.value for word in verbs) + ' ' + ' '.join(
            word.value for word in question_words) + ' '.join(word.value for word in
other_words)
        return self._remove_redundant_punctuation_and_add_question_mark(result)

from algorithm.dto.word.Word import Word
```

```
class Adjective(Word):

    def __init__(self, value, tag, lemma, part_of_speech, number_in_sentence, case,
degree, gender, number):
        super().__init__(value, tag, lemma, part_of_speech, number_in_sentence)
        self.__case = case
        self.__degree = degree
        self.__gender = gender
        self.__number = number

    @property
    def case(self):
        return self.__case

    @case.setter
    def case(self, case):
        self.__case = case

    @property
    def degree(self):
        return self.__degree

    @degree.setter
    def degree(self, degree):
        self.__degree = degree

    @property
    def gender(self):
        return self.__gender
```

```
    @gender.setter
    def gender(self, gender):
        self.__gender = gender

    @property
    def number(self):
        return self.__number

    @number.setter
    def number(self, number):
        self.__number = number

    def __str__(self):
        return '[' \
               + 'value: ' + self._value \
               + ', tag: ' + self._tag.value \
               + ', lemma: ' + self._lemma \
               + ', part_of_speech: ' + self._part_of_speech.value \
               + ', case: ' + self.__case \
               + ', degree: ' + self.__degree \
               + ', gender: ' + self.__gender \
               + ', number: ' + self.__number \
               + ']'

from algorithm.dto.word.Word import Word


class Adverb(Word):
```

```
    def __init__(self, value, tag, lemma, part_of_speech, number_in_sentence, degree):
        super().__init__(value, tag, lemma, part_of_speech, number_in_sentence)
        self.__degree = degree

    @property
    def degree(self):
        return self.__degree

    @degree.setter
    def degree(self, degree):
        self.__degree = degree

    def __str__(self):
        return '[' \
               + 'value: ' + self._value \
               + ', tag: ' + self._tag.value \
               + ', lemma: ' + self._lemma \
               + ', part_of_speech: ' + self._part_of_speech.value \
               + ', degree: ' + self.__degree \
               + ']'

from algorithm.dto.word.Word import Word


class Determinant(Word):

    def __init__(self, value, tag, lemma, part_of_speech, number_in_sentence, case,
number):
        super().__init__(value, tag, lemma, part_of_speech, number_in_sentence)
        self.__case = case
```

```
        self.__number = number

    @property
    def case(self):
        return self.__case

    @case.setter
    def case(self, case):
        self.__case = case

    @property
    def number(self):
        return self.__number

    @number.setter
    def number(self, number):
        self.__number = number

    def __str__(self):
        return '[' \
               + 'value: ' + self._value \
               + ', tag: ' + self._tag.value \
               + ', lemma: ' + self._lemma \
               + ', part_of_speech: ' + self._part_of_speech.value \
               + ', case: ' + self.__case \
               + ', number: ' + self.__number \
               + ']'

from algorithm.dto.word.Word import Word
```

```
class Noun(Word):

    def __init__(self, value, tag, lemma, part_of_speech, number_in_sentence,
animacy, case, gender, number):
        super().__init__(value, tag, lemma, part_of_speech, number_in_sentence)
        self.__animacy = animacy
        self.__case = case
        self.__gender = gender
        self.__number = number

    @property
    def animacy(self):
        return self.__animacy

    @animacy.setter
    def animacy(self, animacy):
        self.__animacy = animacy

    @property
    def case(self):
        return self.__case

    @case.setter
    def case(self, case):
        self.__case = case

    @property
    def gender(self):
        return self.__gender
```

```
    @gender.setter
    def gender(self, gender):
        self.__gender = gender

    @property
    def number(self):
        return self.__number

    @number.setter
    def number(self, number):
        self.__number = number

    def __str__(self):
        return '[' \
               + 'value: ' + self._value \
               + ', tag: ' + self._tag.value \
               + ', lemma: ' + self._lemma \
               + ', part_of_speech: ' + self._part_of_speech.value \
               + ', animacy: ' + self.__animacy \
               + ', case: ' + self.__case \
               + ', gender: ' + self.__gender \
               + ', number: ' + self.__number \
               + ']'

from algorithm.dto.word.Word import Word


class Pronoun(Word):
```

```
    def __init__(self, value, tag, lemma, part_of_speech, number_in_sentence, case,
gender, number, person):
        super().__init__(value, tag, lemma, part_of_speech, number_in_sentence)
        self.__case = case
        self.__gender = gender
        self.__number = number
        self.__person = person

    @property
    def case(self):
        return self.__case

    @case.setter
    def case(self, case):
        self.__case = case

    @property
    def gender(self):
        return self.__gender

    @gender.setter
    def gender(self, gender):
        self.__gender = gender

    @property
    def number(self):
        return self.__number

    @number.setter
    def number(self, number):
```

```
        self.__number = number

    @property
    def person(self):
        return self.__person

    @person.setter
    def person(self, person):
        self.__person = person

    def __str__(self):
        return '[' \
               + 'value: ' + self._value \
               + ', tag: ' + self._tag.value \
               + ', lemma: ' + self._lemma \
               + ', part_of_speech: ' + self._part_of_speech.value \
               + ', case: ' + self.__case \
               + ', gender: ' + self.__gender \
               + ', number: ' + self.__number \
               + ', person: ' + self.__person \
               + ']'

from algorithm.dto.word.Word import Word


class Verb(Word):

    def __init__(self, value, tag, lemma, part_of_speech, number_in_sentence, aspect,
gender, mood, number, tense,
                 verb_form, voice):
```

```
        super().__init__(value, tag, lemma, part_of_speech, number_in_sentence)
        self.__aspect = aspect
        self.__gender = gender
        self.__mood = mood
        self.__number = number
        self.__tense = tense
        self.__verb_form = verb_form
        self.__voice = voice

    @property
    def aspect(self):
        return self.__aspect

    @aspect.setter
    def aspect(self, aspect):
        self.__aspect = aspect

    @property
    def gender(self):
        return self.__gender

    @gender.setter
    def gender(self, gender):
        self.__gender = gender

    @property
    def mood(self):
        return self.__mood

    @mood.setter
```

```
    def mood(self, mood):
        self.__mood = mood

    @property
    def number(self):
        return self.__number

    @number.setter
    def number(self, number):
        self.__number = number

    @property
    def tense(self):
        return self.__tense

    @tense.setter
    def tense(self, tense):
        self.__tense = tense

    @property
    def verb_form(self):
        return self.__verb_form

    @verb_form.setter
    def verb_form(self, verb_form):
        self.__verb_form = verb_form

    @property
    def voice(self):
        return self.__voice
```

```
    @voice.setter
    def voice(self, voice):
        self.__voice = voice

    def __str__(self):
        return '[' \
               + 'value: ' + self._value \
               + ', tag: ' + self._tag.value \
               + ', lemma: ' + self._lemma \
               + ', part_of_speech: ' + self._part_of_speech.value \
               + ', aspect: ' + self.__aspect \
               + ', gender: ' + self.__gender \
               + ', mood: ' + self.__mood \
               + ', number: ' + self.__number \
               + ', tense: ' + self.__tense \
               + ', verb_form: ' + self.__verb_form \
               + ', voice: ' + self.__voice \
               + ']'

from algorithm.service.enum.WordTag import WordTag


class Word:

    def __init__(self, value, tag, lemma, part_of_speech, number_in_sentence):
        self._value = value
        self._tag = tag
        self._base_tag = WordTag[self._tag.value[2:]] if self._tag != WordTag.O else
WordTag.O
```

```
        self._lemma = lemma
        self._part_of_speech = part_of_speech
        self._number_in_sentence = number_in_sentence

    @property
    def value(self):
        return self._value

    @value.setter
    def value(self, value):
        self._value = value

    @property
    def tag(self):
        return self._tag

    @tag.setter
    def tag(self, tag):
        self._tag = tag

    @property
    def base_tag(self):
        return self._base_tag

    @base_tag.setter
    def base_tag(self, base_tag):
        self._base_tag = base_tag

    @property
    def lemma(self):
```

```
        return self._lemma

    @lemma.setter
    def lemma(self, lemma):
        self._lemma = lemma

    @property
    def part_of_speech(self):
        return self._part_of_speech

    @part_of_speech.setter
    def part_of_speech(self, part_of_speech):
        self._part_of_speech = part_of_speech

    @property
    def number_in_sentence(self):
        return self._number_in_sentence

    @number_in_sentence.setter
    def number_in_sentence(self, number_in_sentence):
        self._number_in_sentence = number_in_sentence

    def __str__(self):
        return '[' \
               + 'value: ' + self._value \
               + ', tag: ' + self._tag.value \
               + ', lemma: ' + self._lemma \
               + ', part_of_speech: ' + self._part_of_speech.value \
               + ']'
```

```
class Question:

    def __init__(self, question, answer, sentence, tag):
        self.__question = question
        self.__answer = answer
        self.__sentence = sentence
        self.__tag = tag
        self.__options = list()

    @property
    def question(self):
        return self.__question

    @question.setter
    def question(self, question):
        self.__question = question

    @property
    def answer(self):
        return self.__answer

    @answer.setter
    def answer(self, answer):
        self.__answer = answer

    @property
    def sentence(self):
        return self.__sentence

    @sentence.setter
```

```
    def sentence(self, sentence):
        self.__sentence = sentence

    @property
    def tag(self):
        return self.__tag

    @tag.setter
    def tag(self, tag):
        self.__tag = tag

    @property
    def options(self):
        return self.__options

    @options.setter
    def options(self, options):
        self.__options = options

    def __str__(self):
        return '[' \
               + 'question: "' + self.__question \
               + '", answer: "' + ' '.join(self.__answer) \
               + '", sentence: "' + str(self.__sentence) \
               + '", options: "' + ' '.join(self.__options) \
               + '"]'


class Sentence:

    def __init__(self, words):
```

```
        self.__words = words
        self.__fill_questions = list()
        self.__answer_questions = list()

    @property
    def words(self):
        return self.__words

    @words.setter
    def words(self, words):
        self.__words = words

    @property
    def fill_questions(self):
        return self.__fill_questions

    @fill_questions.setter
    def fill_questions(self, fill_questions):
        self.__fill_questions = fill_questions

    @property
    def answer_questions(self):
        return self.__answer_questions

    @answer_questions.setter
    def answer_questions(self, answer_questions):
        self.__answer_questions = answer_questions

    def __str__(self):
        return ' '.join(word.value for word in self.__words)
```

```
import random
import re
from abc import ABC

from algorithm.dto.Question import Question
from algorithm.service.enum.WordTag import WordTag


class AbstractQuestionService(ABC):
    __fill_question_gap = '_____'

    def __init__(self, base_tag, b_tag, i_tag):
        self._base_tag = base_tag
        self._b_tag = b_tag
        self._i_tag = i_tag
        self._fill_phrase = ''

    def generate_questions(self, sentences, text_answers, similar_answers):
        for sentence in sentences:
            self._generate_fill_questions(sentence, text_answers, similar_answers)
            if len(self._extract_words_by_tag(sentence.words, self._b_tag)) == 1:
                self._generate_special_questions(sentence)

    def _generate_fill_questions(self, sentence, text_answers, similar_answers):
        words = sentence.words
        words_by_base_tag = self._extract_words_by_base_tag(words, self._base_tag)
        answer, possible_answers =
self._extract_fill_question_answers(words_by_base_tag, text_answers,
similar_answers)
```

```
        question = Question(self._convert_to_fill_question_string_form(words,
self._fill_phrase), answer, sentence,
                            self._base_tag)
        question.options = possible_answers
        sentence.fill_questions.append(question)

    def _convert_to_fill_question_string_form(self, words, fill_phrase):
        result = fill_phrase
        for word in words:
            if word.tag == self._b_tag:
                result = result + ' ' + self.__fill_question_gap
            elif word.tag != self._i_tag:
                result = result + ' ' + word.value
        return result

    def _extract_fill_question_answers(self, key_words, text_answers,
similar_answers):
        result = []
        answer = ''
        for word in key_words:
            if word.tag == self._b_tag:
                if answer:
                    result.append(answer + ';')
                answer = word.value
            else:
                answer = answer + ' ' + word.value
        result.append(answer)
        if len(result) == 1:
            return result, self._generate_possible_answers(key_words, text_answers,
similar_answers)
```

```
        else:
            return result, []

    def _generate_possible_answers(self, key_words, text_answers, similar_answers):
        result = []
        main_word = key_words[0]
        answers_from_text = text_answers[self._base_tag]
        answers_from_text.remove(key_words)
        for answer in random.sample(answers_from_text, len(answers_from_text) if
len(answers_from_text) <= 2 else 2):
            result.append(self._extract_special_question_answers(answer))
        for answer in similar_answers[main_word.lemma]:
            if '_' in answer:
                parts = answer.split('_')
                if main_word.part_of_speech.value == parts[1]:
                    result.append(parts[0])
            else:
                result.append(answer)
            if len(result) == 3:
                break
        return result

    def _generate_special_questions(self, sentence):
        pass

    def _convert_to_special_question_string_form(self, question_words, verbs,
other_words):
        pass

    @staticmethod
```

```
    def _extract_special_question_answers(answer_words):
        return [' '.join(word.value for word in answer_words)]

    @staticmethod
    def _extract_words_by_base_tag(words, tag):
        return list(filter(lambda w: w.base_tag == tag, words))

    @staticmethod
    def _extract_words_by_tag(words, tag):
        return list(filter(lambda w: w.tag == tag, words))

    @staticmethod
    def _split_sentence_by_punctuation(words):
        result = []
        for i, word in enumerate(words):
            if word.tag == WordTag.O and re.fullmatch(r'[—\-,.!?]', word.value):
                result.append(words[:i + 1])
        return result

    @staticmethod
    def _extract_words_by_parts_of_speech(words, part_of_speech):
        return list(filter(lambda w: w.part_of_speech in part_of_speech, words))

    @staticmethod
    def _extract_remaining_words(words, words_do_delete):
        result = words
        for word_to_delete_list in words_do_delete:
            result = [w for w in result if w not in word_to_delete_list]
        return result
```

```
    @staticmethod
    def _remove_redundant_punctuation_and_add_question_mark(string):
        reversed_string = string[::-1]
        for char in reversed_string:
            if re.fullmatch(r'[—\-,.!? ]', char):
                string = string[:-1]
            else:
                return string + '?'
        raise ValueError('Error during clearing redundant punctuation from question')


import logging

from algorithm.methods.filtersentences import filter_sentences_by_tags
from algorithm.service.enum.WordTag import WordTag
from algorithm.service.model.GensimModelService import GensimModelService
from algorithm.service.question.type.DateQuestionService import
DateQuestionService
from algorithm.service.question.type.GpeQuestionService import
GpeQuestionService
from algorithm.service.question.type.LocationQuestionService import
LocationQuestionService
from algorithm.service.question.type.PersonQuestionService import
PersonQuestionService
from algorithm.service.text.TextProcessorService import TextProcessorService


class GenerateQuestionsService:
    __textProcessorService = TextProcessorService()
    __gensimModelService = GensimModelService()
```

```
    __question_services = {
        WordTag.PERSON: PersonQuestionService(WordTag.PERSON,
WordTag.B_PERSON, WordTag.I_PERSON),
        WordTag.GPE: GpeQuestionService(WordTag.GPE, WordTag.B_GPE,
WordTag.I_GPE),
        WordTag.LOC: LocationQuestionService(WordTag.LOC, WordTag.B_LOC,
WordTag.I_LOC),
        WordTag.DATE: DateQuestionService(WordTag.DATE, WordTag.B_DATE,
WordTag.I_DATE)
    }

    def generate_questions(self, text):
        sentences = self.__textProcessorService.split_text_to_sentences(text)
        sentences = self.__textProcessorService.replace_pronouns(sentences)
        possible_text_answers = self.__search_possible_answers_in_text(sentences)
        possible_similar_answers = self.__gensimModelService.analyze(sentences)
        sentences_by_tags = filter_sentences_by_tags(sentences)
        for tag, sentences_by_tag in sentences_by_tags.items():
            if tag in self.__question_services:
                self.__question_services[tag].generate_questions(sentences_by_tag,
possible_text_answers,
                                                 possible_similar_answers)

        self.__log_questions(sentences)

    def __search_possible_answers_in_text(self, sentences):
        result = {}
        for sentence in sentences:
            possible_answer = []
            for word in sentence.words:
```

```
            if word.base_tag != WordTag.O:
                possible_answer.append(word)
            elif len(possible_answer) != 0:
                result = self.__put_in_result(result, possible_answer)
                possible_answer = []
        if len(possible_answer) != 0:
            result = self.__put_in_result(result, possible_answer)
    return result

@staticmethod
def __put_in_result(result, possible_answer):
    tag = possible_answer[0].base_tag
    if tag not in result:
        result[tag] = []
    result[tag].append(possible_answer)
    return result

@staticmethod
def __log_questions(sentences):
    logging.info('Generated questions:\n')
    for sentence in sentences:
        logging.info(str(sentence))

        fill_questions = sentence.fill_questions
        if fill_questions:
            logging.info('Word filling questions: ')
            for fill_question in fill_questions:
                logging.info(fill_question.question + ' , answer: ' + '
'.join(fill_question.answer))
```

```
            answer_questions = sentence.answer_questions
            if answer_questions:
                logging.info('Questions with special word: ')
                for answer_question in answer_questions:
                    logging.info(answer_question.question + ', answer: ' + '
'.join(answer_question.answer))

            logging.info('')

import logging

from rusenttokenize import ru_sent_tokenize

from algorithm.dto.word.Noun import Noun
from algorithm.methods.extractsentences import extract
from algorithm.service.enum.PartOfSpeech import PartOfSpeech
from algorithm.service.model.MorphologicalModelService import
MorphologicalModelService
from algorithm.service.model.NerModelService import NerModelService


class TextProcessorService:
    __nerModelService = NerModelService()
    __morphologicalModelService = MorphologicalModelService()

    def split_text_to_sentences(self, text):
        string_sentences = self.__tokenize(text)
        ner_sentences = self.__nerModelService.analyze(string_sentences)
        morphological_sentences = self.__morphologicalModelService.analyze(
            self.__convert_ner_sentence_words_into_new_sentences(ner_sentences))
```

```
        result = extract(ner_sentences, morphological_sentences)
        self.__log_converted_sentences(result)
        return result

    @staticmethod
    def __tokenize(text):
        return ru_sent_tokenize(text)

    @staticmethod
    def __convert_ner_sentence_words_into_new_sentences(ner_sentences):
        result = []
        for pair in ner_sentences:
            result.append(' '.join(pair[0]))
        return result

    @staticmethod
    def __log_converted_sentences(sentences):
        for sentence in sentences:
            logging.info(str(sentence))
            for word in sentence.words:
                logging.info(word)
            logging.info('')

    def replace_pronouns(self, sentences):
        for i, sentence in enumerate(sentences):
            for j, word in enumerate(list(sentence.words)):
                if word.part_of_speech == PartOfSpeech.PRON:
                    words_for_replace = self.__process_replace(word, sentences[i - 1])
                    before = sentence.words[:j]
                    after = sentence.words[j + 1:]
```

```
                sentence.words = before
                sentence.words.extend(words_for_replace)
                sentence.words.extend(after)
        return sentences

    @staticmethod
    def __process_replace(pron, previous_sentence):
        result = []
        words = list(filter(lambda w: isinstance(w, Noun), previous_sentence.words))
        for word in words:
            if word.gender == pron.gender and word.number == pron.number:
                result.append(word)
        filtered_by_case = list(filter(lambda w: w.case == pron.case, result))
        if filtered_by_case:
            return filtered_by_case
        return result

import logging

from deeppavlov import build_model, configs


def load_ner_model():
    model = build_model(configs.ner.ner_ontonotes_bert_mult, download=True)
    logging.info("NER model loaded successfully")
    return model


def load_morphological_model():
```

```
    model = build_model(configs.morpho_tagger.BERT.morpho_syntagrus_bert,
download=True)
    logging.info("Morphological model loaded successfully")
    return model


import logging

import gensim.downloader as api



def load_gensim_model():
    model = api.load("word2vec-corpora-300")
    logging.info("Gensim model loaded successfully")
    return model



def load_gensim_number_model():
    model = api.load("glove-wiki-gigaword-50")
    logging.info("Gensim num model loaded successfully")
    return model
```

Факультет інформатики та обчислювальної техніки

Кафедра автоматизованих систем обробки інформації і управління

“ЗАТВЕРДЖЕНО”

В.о. завідувача кафедри

_____ Олександр ПАВЛОВ

“___” _____ 2020 р.

ВЕБ-ЗАСТОСУВАННЯ ДЛЯ ГЕНЕРАЦІЇ ПИТАНЬ ЗА ТЕКСТОВИМИ ДАНИМИ

Програма та методика тестування

КПІ.ІП-6319.045440.04.51

“ПОГОДЖЕНО”

Керівник проєкту:

_____ Іванова Л.М.

Нормоконтроль:

_____ К.І. Ліщук

Виконавець:

_____О.О. Міщенко

Київ – 2020 року

ЗМІСТ

1 ОБ’ЄКТ ВИПРОБУВАНЬ

Об’єктом випробувань, що описані в даному документі, є веб-застосування, що призначене для аналізу тексту та автоматичної генерації тестових питань по іменованим сутностям. В рамках даного випробування проводиться дослідження серверної та клієнтська частини версії v1.3 веб застосунку.

2 МЕТА ТЕСТУВАННЯ

Метою тестування є:

– визначити ступінь відповідності реальної поведінки створеного веб застосунку відповідно до очікуваній поведінці, що описана у функціональних та нефункціональних вимогах до цього програмного продукту;

– прийняти рішення про придатність або непридатність поточної версії розробленого сервісу до промислової експлуатації.

3 МЕТОДИ ТЕСТУВАННЯ

Тестування проводиться на основі заведення тестових прецедентів (test case) для кожної функції, яка буде тестуватися. Опис тестового прецеденту включає в себе:

- унікальний ідентифікатор;
- текстовий опис;
- передумови;
- постумови;
- основний сценарій проведення тестування прецеденту;
- очікуваний результат;
- отриманий результат;
- дату створення прецеденту.

Тестування має проводиться для кожного тестового прецеденту. Спосіб ведення обліку результатів тестування за кожним тестовим прецедентом не регламентується. Вибір способу покладено на тестувальника розробленого програмного забезпечення.

4 ЗАСОБИ ТА ПОРЯДОК ТЕСТУВАННЯ

Тестування розробленого програмного продукту має проводиться з використанням наступних програмних та технічних засобів.

Програмні засоби:

– операційна система Windows або Linux;

– браузер Google Chrome версії 80.

У склад технічних засобів входить персональний комп’ютер, з наступними характеристиками:

– процесор Intel Core i5 не менше ГГц – 1.5;

– оперативну пам’ять не менше 1 Гб;

– жорсткий або твердотільний накопичувач не менше 10 Гб.

Для проведення випробувань розробник даного програмного забезпечення надає інсталяційну версію розробленого продукту.

Факультет інформатики та обчислювальної техніки

Кафедра автоматизованих систем обробки інформації і управління

“ЗАТВЕРДЖЕНО”

В.о. завідувача кафедри

_____ Олександр ПАВЛОВ

“___” _____ 2020 р.

ВЕБ ЗАСТОСУВАННЯ ДЛЯ ГЕНЕРАЦІЇ ПИТАНЬ ЗА ТЕКСТОВИМИ ДАНИМИ

Керівництво користувача

КПІ.ІП-6319.045440.05.34

“ПОГОДЖЕНО”

Керівник проєкту:

_____ Л.М. Іванова

Нормоконтроль:

_____ К.І. Ліщук

Виконавець:

_____О.О. Міщенко

Київ – 2020 року

ЗМІСТ

1 ПРИЗНАЧЕННЯ ПРОГРАМИ

1.1 Функціональне призначення

Якість програмного забезпечення - це сукупність певних характеристик розробленого веб застосування, що відповідають до його можливості задовольняти встановлені та очікувані функціональні вимоги. Для проведення випробувань та визначення відповідності реальної поведінки системи описаним вимогам був розроблений план тестування відповідно до міжнародного стандарту IEEE 892-2008.

Функціональним призначенням розробленого веб застосування є надання можливості користувачу автоматичної генерації тестових питань на основі текстових даних.

1.2 Експлуатаційне призначення

Програма призначена для експлуатації людьми, що працюють у сфері освіти та постійно мають контролювати якість засвоєного матеріалу у учнів у вигляді тестів.

1.3 Склад функцій

Програма забезпечує можливість виконання наведених нижче функцій:

- функції реєстрації в системі;
- функції авторизації в системі;
- функції введення тексту для опрацювання;
- функції завантаження і конвертації документів формату txt, doc та pdf;
- функції вибору речень для формування питань;
- функції вибору ключових слів для формування питань;
- функції вибору тегів для формування питань;

- функції генерації питань по тексту;
- функції експорту результату у форматі txt, doc та pdf документів;
- функції перегляду історії опрацьованих текстів.

2 УМОВИ ВИКОНАННЯ ПРОГРАМИ

2.1 Кліматичні умови експлуатації

Кліматичні умови експлуатації повинні повністю задовольняти вимогам, що висуваються до технічних засобів, необхідних для експлуатації програмного забезпечення і роботи усіх зазначених функцій.

2.2 Мінімальний склад технічних засобів

До мінімального складу технічних засобів повинен входити комп'ютер з наступними характеристиками:

- процесор Intel Core i5 не менше ГГц – 1.5;
- оперативну пам’ять не менше 1 Гб;
- жорсткий або твердотільний накопичувач не менше 10 Гб.

2.3 Мінімальний склад програмних засобів

До мінімального складу програмних засобів повинен входити програми з наступними характеристиками:

- операційна система Linux/Windows;
- postgreSQL версії 12.0;
- браузер Google Chrome версії 80.

3 ВИКОНАННЯ ПРОГРАМИ

3.1 Початок роботи

Для початку роботи з програмним забезпеченням, користувач повинен відкрити браузер та ввести у адресний рядок домен серверу з розгорнутим веб застосунком. Для того щоб дізнатися домен серверу, на якому розгорнуто створене веб застосування, користувач має звернутись до програміста, що проводив розгортання даного сервісу.

3.2 Використання програми

Перш ніж використовувати програму, користувачу необхідно мати підготовлений текст з наявністю іменованих сутностей у ньому в одному с наступних форматів: звичайний текст, doc, pdf або txt.

Після визначення тексту, на базі якого будуть генеруватися питання, користувач вносить його до веб застосунку.

3.2.1 Реєстрації у системі

Для реєстрації у системі, необхідно:

– перейти до основної сторінки веб застосування;

– натиснути кнопку “Sign up”;

– у формі авторизації, ввести емайл, нік та пароль до нового облікового запису;

– натиснути кнопку “Sign up”;

У разі, якщо кнопка перебуває у нерозблокованому вигляді, необхідно виконати вказівки програми, можливо користувач з таким емейлом або ніком вже існує. Для вирішення проблеми користувачеві треба змінити вказані поля та спробувати ще раз.

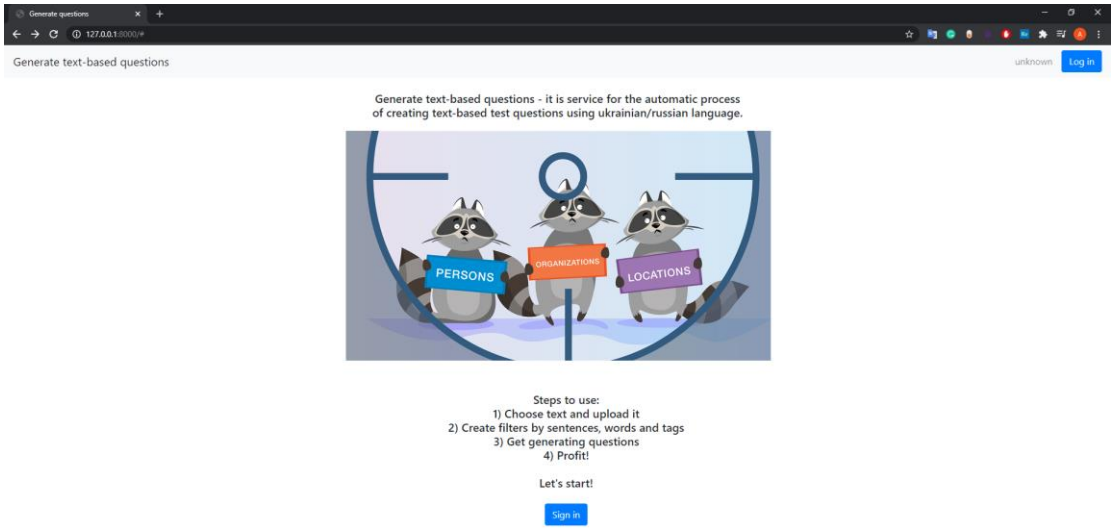


Рисунок 3.1 – Відкрита основна сторінка веб-застосунку

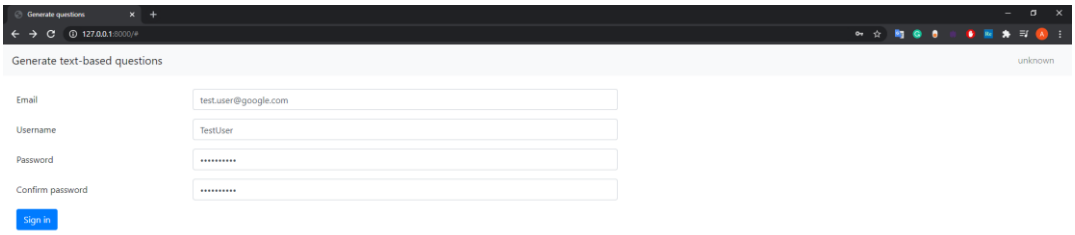


Рисунок 3.2 – Приклад заповненої форми реєстрації

3.2.2 Авторизація у системі

Для авторизація у системі, необхідно:

– перейти до основної сторінки веб застосування;

– натиснути кнопку “Log in”;

– у формі авторизації, ввести нік та пароль до існуючого облікового запису;

– натиснути кнопку “Log in”.

У разі, якщо кнопка перебуває у нерозблокованому вигляді, необхідно виконати вказівки програми, можливо користувач з таким емейлом або ніком не існує. Для вирішення проблеми користувачеві треба змінити вказані поля та спробувати ще раз.

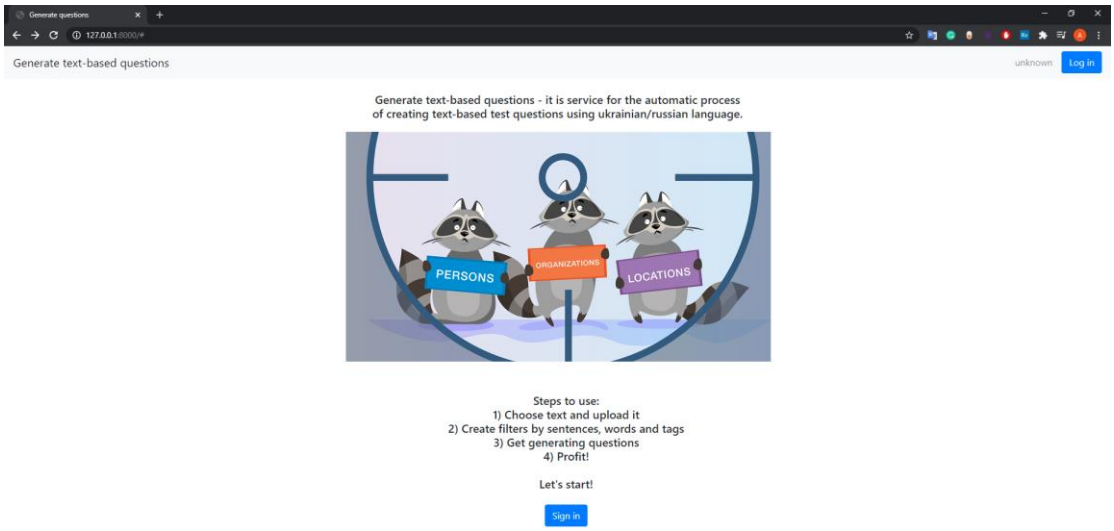


Рисунок 3.3 – Відкрита основна сторінка веб-застосунку

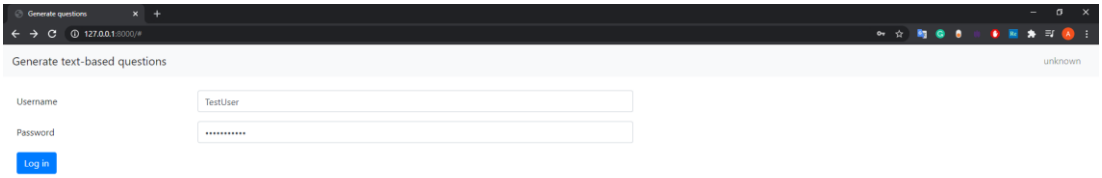


Рисунок 3.4 – Приклад заповненої форми авторизації

3.2.3 Введення тексту для опрацювання

Для введення тексту для опрацювання необхідно:

– Перейти до сторінки “Create” веб застосування.

– Додати текст до форми.

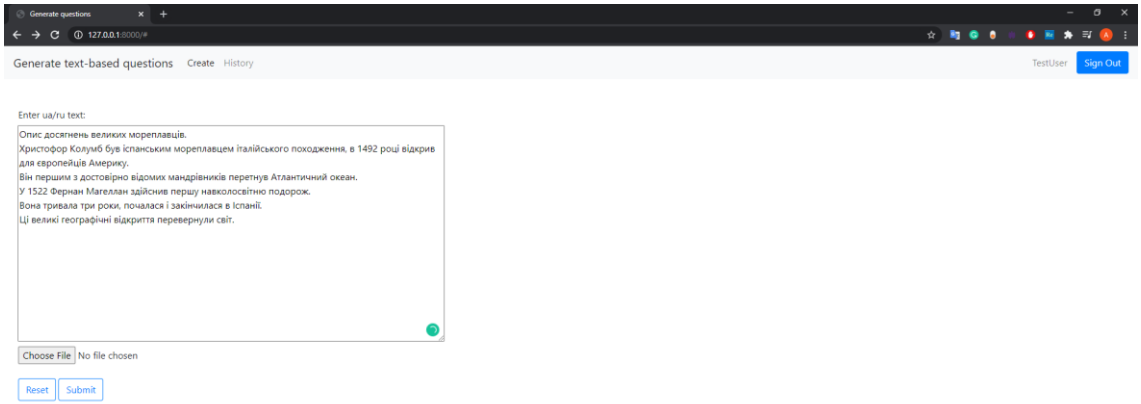


Рисунок 3.5 – Сторінка “Create” з вставленим текстом

3.2.4 Завантаження тексту у форматах txt, doc, pdf

Для завантаження тексту у форматах txt, doc, pdf необхідно:

– перейти до сторінки “Create” веб застосування;

– додати текст до форми Натиснути кнопку “Choose file”;

– обрати файл відповідного формату та натиснути кнопку “Open”.

Текст конвертується до форми автоматично.

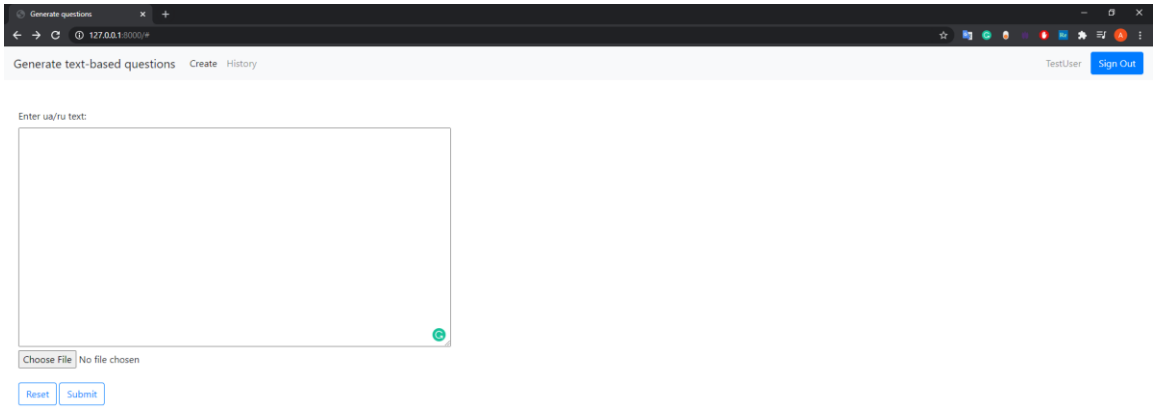


Рисунок 3.6 – Сторінка “Create”

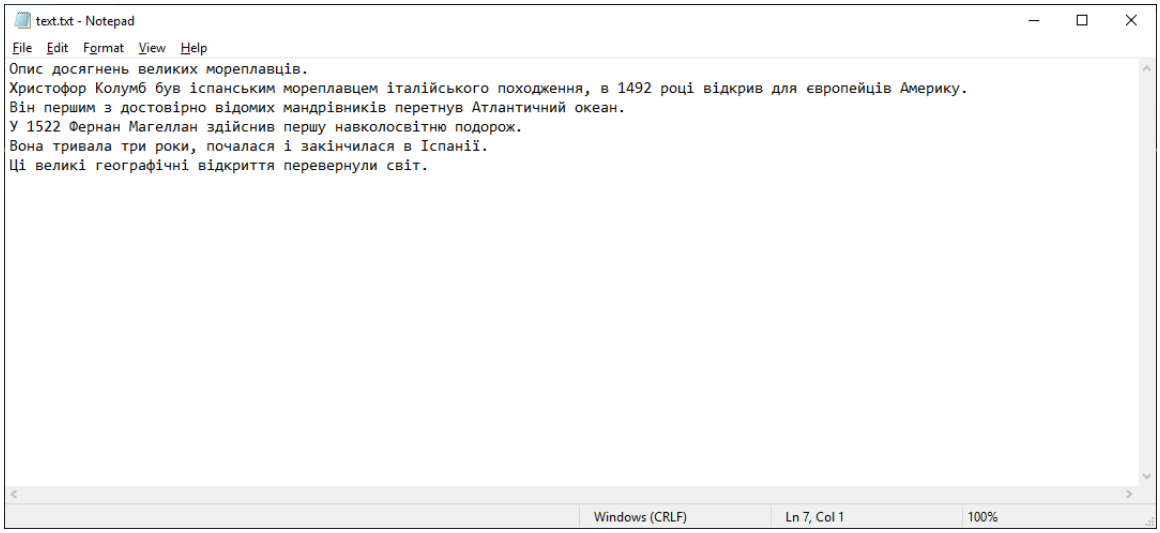


Рисунок 3.7 – Приклад файлу формату txt

Рисунок 3.8 – Приклад файлу формату doc

Рисунок 3.9 – Приклад файлу формату pdf

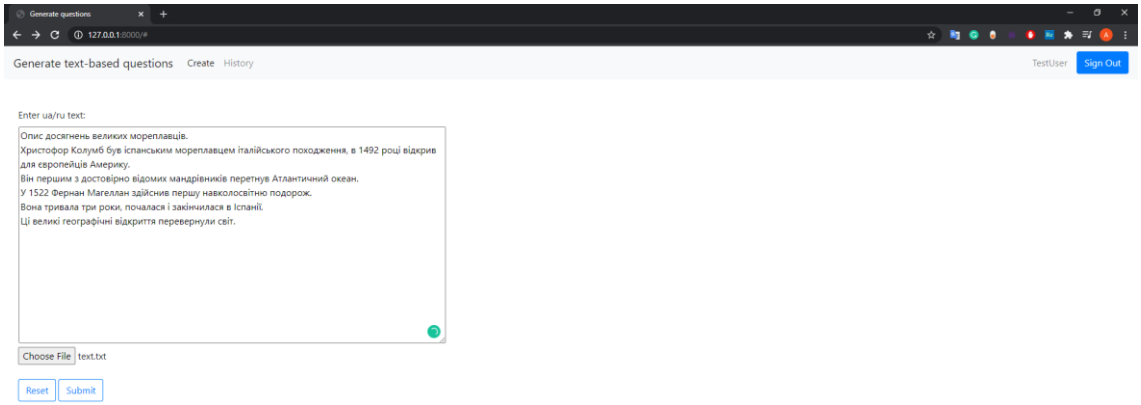


Рисунок 3.10 – Приклад конвертованого файлу формату txt

3.2.5 Вибір речень для генерації питань

Для вибору речень для генерації питань необхідно:

– перейти до сторінки “Create” веб застосування;

– додати текст до форми;

– натиснути кнопку “Submit”;

– у створеної формі з речень обрати ті, до котрих будуть генеруватися питання.

Для вибору речень для генерації питань необхідно застосовувати checkbox біля кожного речення. Варто відзначити, що неможливо ввімкнути checkbox біля речення, на основі якого неможливо згенерувати питання.

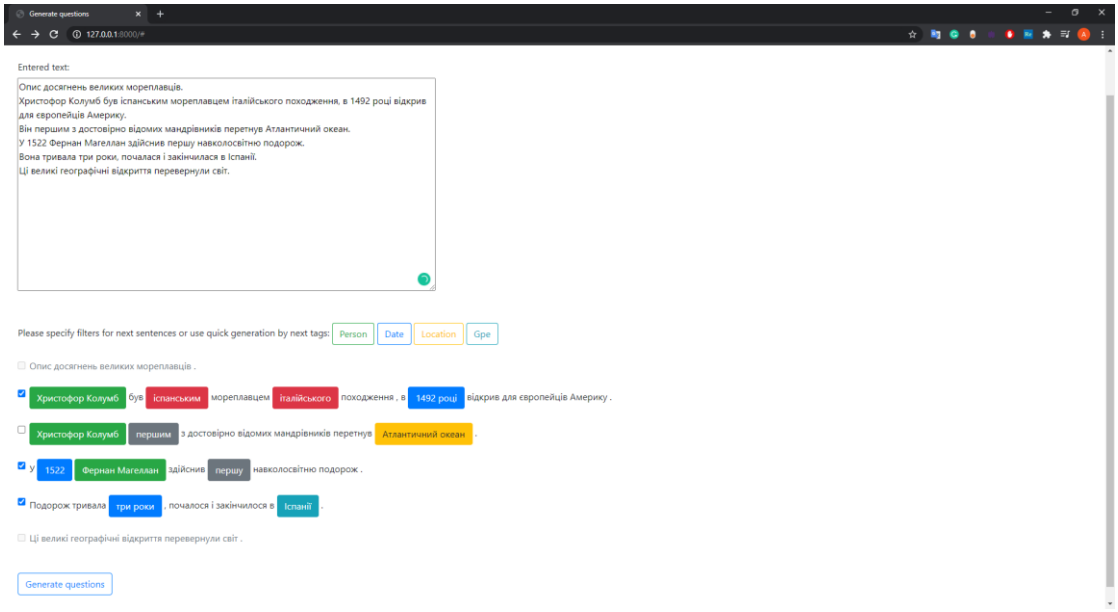


Рисунок 3.11 – Сторінка “Create” з встановленими фільтрами на речення

3.2.6 Вибір ключових слів для генерації питань

Для вибору ключових слів для генерації питань необхідно:

– перейти до сторінки “Create” веб застосування;

– додати текст до форми;

– натиснути кнопку “Submit”;

– у створеної формі з речень обрати ті ключові слова, до котрих будуть генеруватися питання.

Для вибору ключових слів для генерації питань у реченнях необхідно натискати на кнопку конкретної іменованої сутності у реченні, щоб вона придбала насичений відтінок. Варто відзначити, що за замовченням усі слова використовуються для генерації питань.

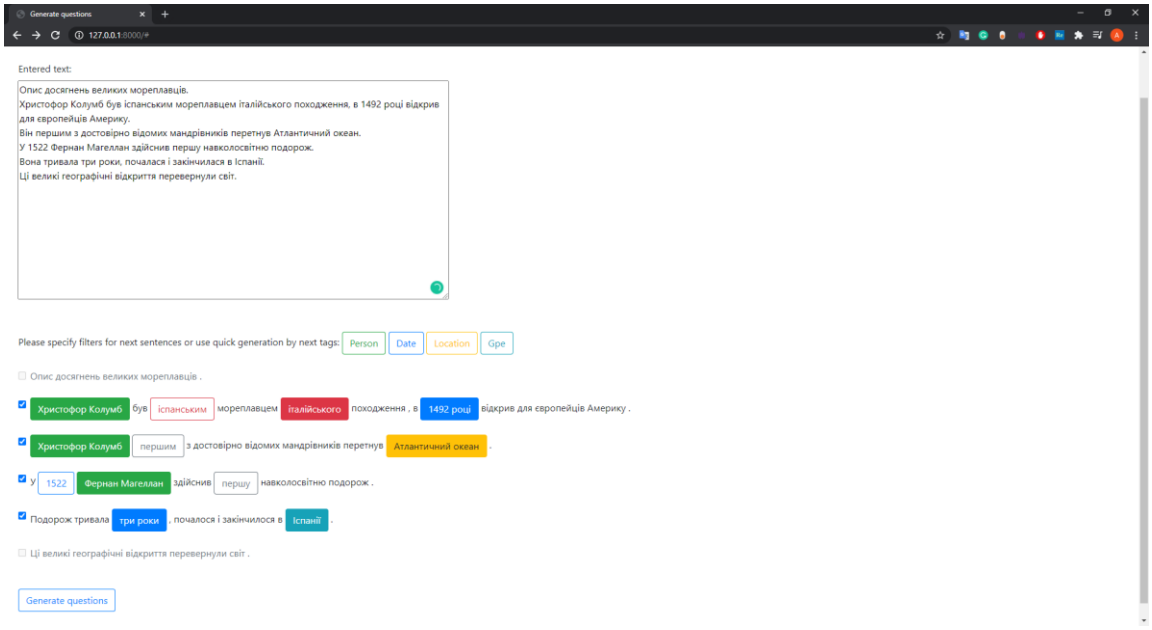


Рисунок 3.12 – Сторінка “Create” з встановленими фільтрами на ключові слова

3.2.7 Вибір тегів для генерації питань

Для вибору тегів слів для генерації питань необхідно:

– перейти до сторінки “Create” веб застосування;

– додати текст до форми;

– натиснути кнопку “Submit”;

– у створеної формі з реченнями обрати теги, за якими будуть генеруватися питання у всьому тексті.

Для вибору тегів для генерації питань у тексті необхідно натискати на кнопки конкретного тега, які розташовані вище згенерованих речень, щоб вони придбали насичений відтінок. Варто відзначити, що за замовченням усі тегі ввімкнені. Якщо було обрано будь-який тег, то алгоритм генерує питання по всім іменованим сутностям цього тега, незважаючи на інші фільтри користувача.

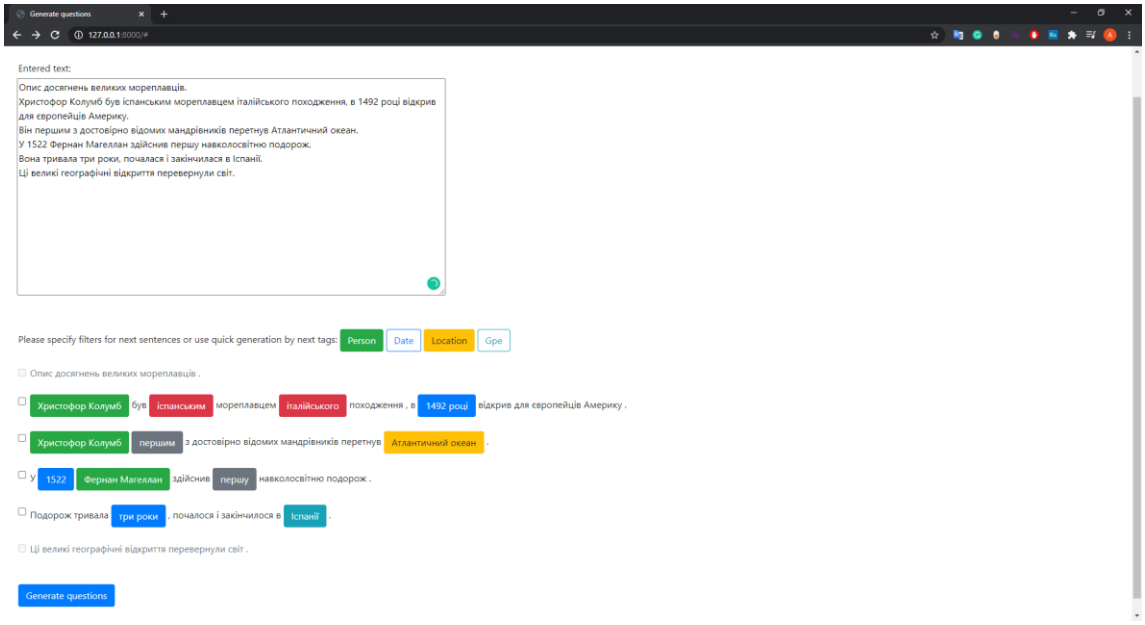


Рисунок 3.13 – Сторінка “Create” з встановленими фільтрами за тегами

3.2.8 Генерація питань

Для генерації питань необхідно:

– перейти до сторінки “Create” веб застосування;

– додати текст до форми;

– натиснути кнопку “Submit”;

– у створеної формі з реченнями обрати фільтри, за якими будуть генеруватися питання;

– натиснути кнопку “Generate questions”.

Рисунок 3.14 – Сторінка “Create” з створеними тестовими питаннями

3.2.9 Експорт згенерованих питань

Для експорту згенерованих генерації необхідно:

– перейти до сторінки “Create” веб застосування;

– додати текст до форми;

– натиснути кнопку “Submit”;

– у створеної формі з реченнями обрати фільтри, за якими будуть генеруватися питання;

– натиснути кнопку “Generate questions”;

– натиснути кнопку “Download txt”, “Download doc” або “Download pdf”.

Рисунок 3.15 – Приклад експорту у txt файл

Рисунок 3.16 – Приклад експорту у doc файл

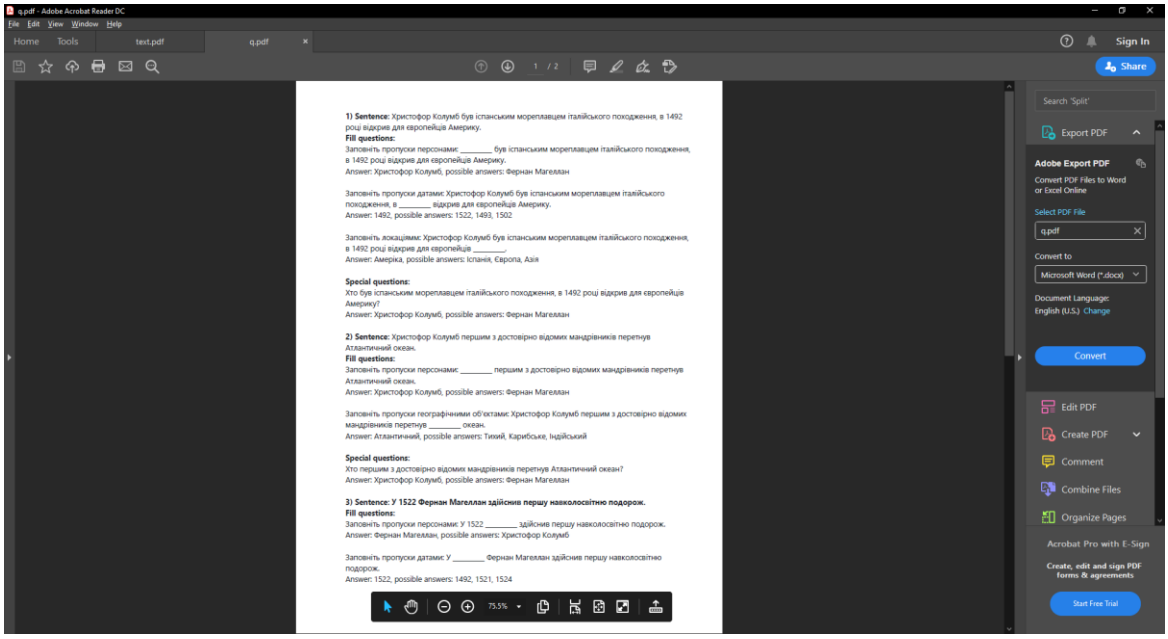


Рисунок 3.17 – Приклад експорту у pdf файл

3.2.10 Перегляд історії користування

Для перегляду історії користування необхідно:

– авторизуватися;

– перейти до сторінки “History” веб застосування.

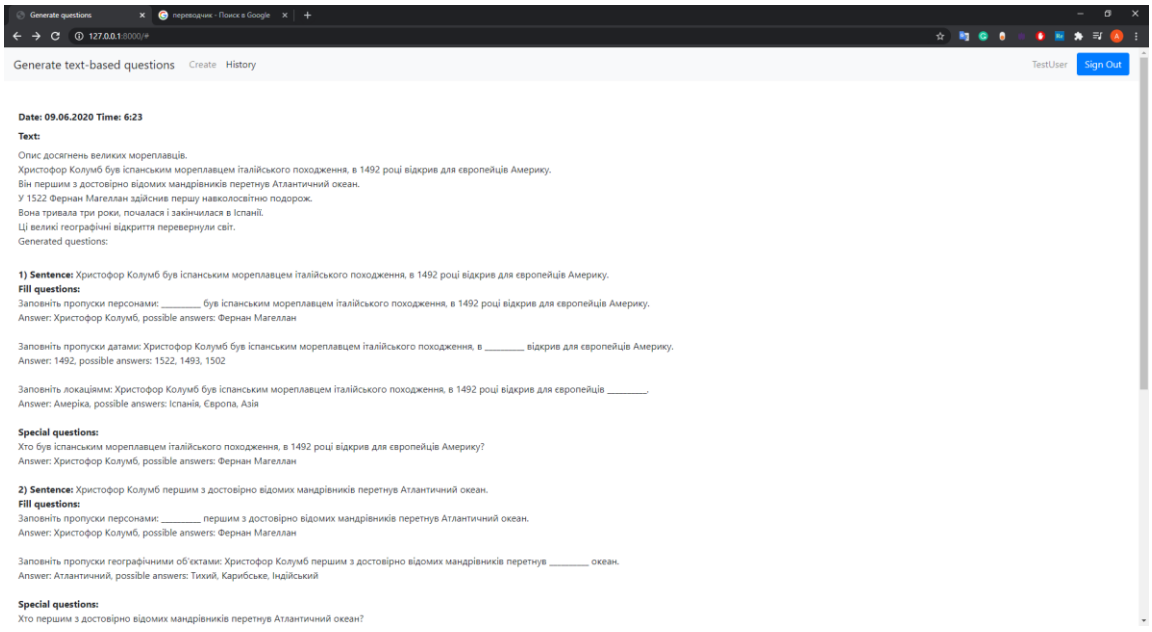


Рисунок 3.18 – Сторінка “History”

3.3 Завершення роботи

Для завершення роботи з програмним забезпеченням, користувач повинен натиснути кнопку “Sign out”.

Факультет інформатики та обчислювальної техніки

Кафедра автоматизованих систем обробки інформації і управління

“ЗАТВЕРДЖЕНО”

В.о. завідувача кафедри

_____ Олександр ПАВЛОВ

“___” _____ 2020 р.

ВЕБ ЗАСТОСУВАННЯ ДЛЯ ГЕНЕРАЦІЇ ПИТАНЬ ЗА ТЕКСТОВИМИ ДАНИМИ

Графічні матеріали

КПІ.ІП-6319.045440.06.99

“ПОГОДЖЕНО”

Керівник проєкту:

_____ Л.М. Іванова

Нормоконтроль:

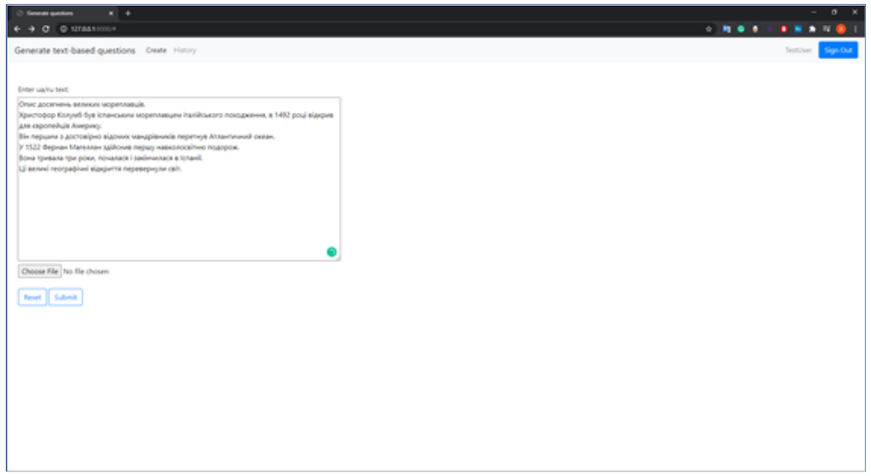
_____ К.І. Ліщук

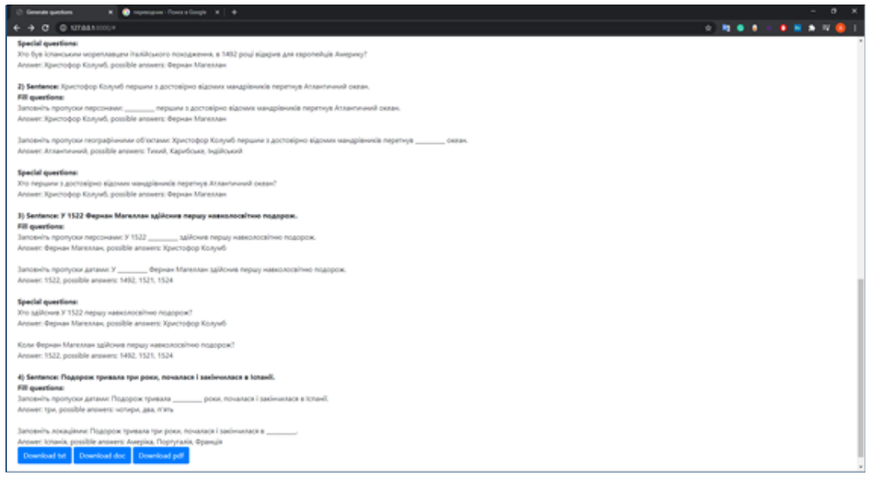
Виконавець:

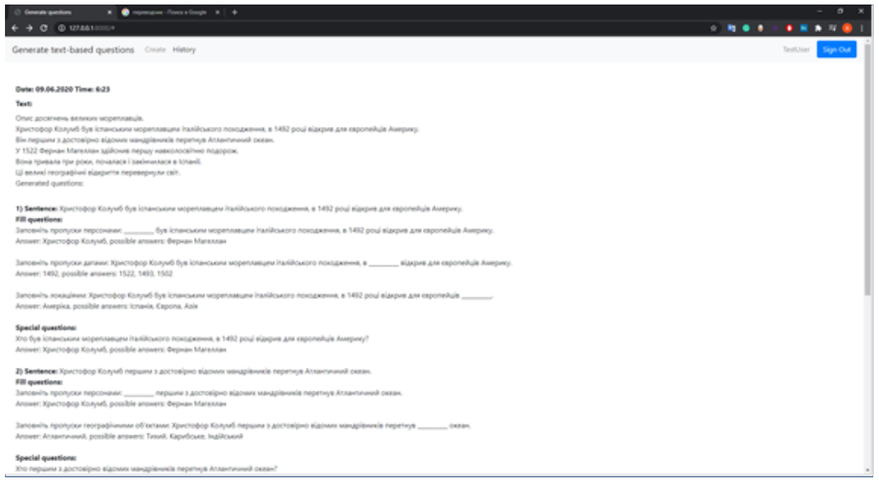
_____О.О. Міщенко

Київ – 2020 року

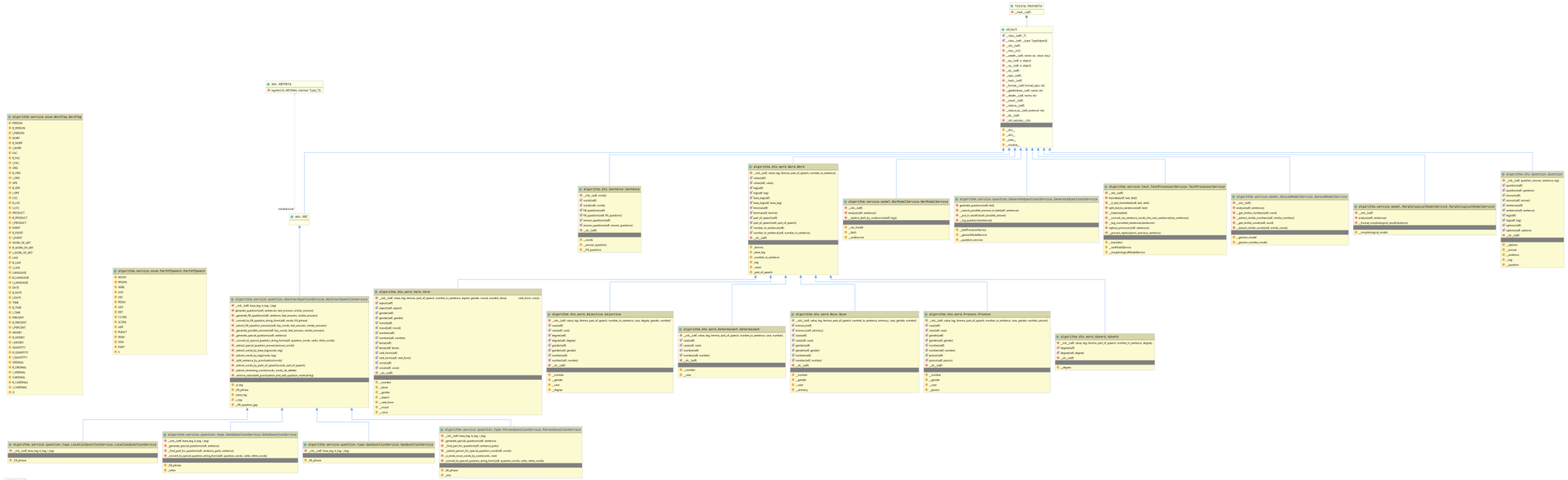
					КПІ.ІП-6319.045440.06.99.СС			
					Схема структурна варіантів використання	Літера	Маса	Масштаб
Зм.	Арк.	№ документа	Підпис	Дата				
Розробив		Міщенко О.О.				Аркуш	Аркушів	
Перевірив		Іванова Л.М.						
І. кон.								
Н. кон.		Ліщук К.І.			Веб застосування для генерації питань за текстовими даними	КПІ ім.Ігоря Сікорського Кафедра АСОІУ гр. ІП-63		
Затвердив		Іванова Л.М.						







					КПІ.ІП-6319.045440.07.99.КЕ		
					Креслення вигляду екранних форм	Літера	Маса / Масштаб
Зм.	Арк.	№ документа	Підпис	Дата			
Розробив		Міщенко О.О.				Аркуш	Аркушів
Перевірив		Іванова Л.М.					
Т. кон.							
					Веб застосування для генерації питань за текстовими даними	КПІ ім.Ігоря Сікорського Кафедра АСОІУ гр. ІП-63	
Н. кон.		Ліщук К.І.					
Затвердив		Іванова Л.М.					



					КПІ.ІП-6319.045440.08.99.СС
Зм.	Арк.	№ документа	Підпис	Дата	
Розробив		Міщенко О.О.			Схема структурна класів програмного забезпечення
Перевірив		Іванова Л.М.			
Н. кон.					
Н. кон.		Ліщук К.І.			Веб застосування для генерації питань за текстовими даними
Затвердив		Іванова Л.М.			

Літера	Маса	Масштаб
Аркуш	Аркушів	
КПІ ім.Ігоря Сікорського Кафедра АСОІУ гр. ІП-63		